University of Zurich[UZH]

Department of Informatics

FlexiSketch: Combining Free-Form Sketching with Lightweight Metamodeling

Dissertation submitted to the Faculty of Business, Economics and Informatics
of the University of Zurich

to obtain the degree of
Doktor der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, PhD)

presented by
Dustin Wüest
from Udligenswil LU, Switzerland

approved in April 2016

at the request of
Prof. Dr. Martin Glinz
Prof. Dr. André van der Hoek

University of Zurich[UZH]

The Faculty of Business, Economics and Informatics of the University of Zurich hereby authorizes the printing of this dissertation, without indicating an opinion of the views expressed in the work.

Zurich, April 06, 2016

Chairwoman of the Doctoral Board: Prof. Dr. Elaine M. Huang

Making the simple complicated is commonplace;
making the complicated simple, awesomely simple, that's creativity.

—Charles Mingus

(Musician, 1922-1979)

Nothing is lost, nothing is created,
everything is transformed.

—Antoine Laurent de Lavoisier

(Chemist & lawyer, 1743-1794)

Acknowledgement

First and foremost I would like to thank Prof. Dr. Martin Glinz for his valuable advice and help throughout the years and for providing me with the opportunity to conduct my PhD studies in his research group. I was very lucky to have him as my advisor.

I would also like to thank Prof. Dr. André van der Hoek for the always inspiring talks and discussions, and for agreeing to co-advise my thesis.

Soon after starting my PhD studies, I realized that I was working in a fantastic environment formed by a research group that consists of very nice and kind fellow researchers: I would like to thank Dr. Norbert Seyff for being a deputy advisor, helping to write papers, and for giving much advice regarding my current and future work. I am also very happy about all the constructive feedback and great help that I received from my research colleagues. My sincerest thanks go to Samuel Fricker, Cédric Jeanneret, Eya Ben Charrada, Arun Mukhija, Tobias Reinhard, Reinhard Stoiber, Irina Todoran Koitz, Anne Koziolek, Parisa Ghazi, Martina Z. Huber, Sofija

Hotomski, Melanie Stade, and Emitzá Guzmán. Without their help, my PhD studies would have been a lot harder and much more boring. I had such an amazing time thanks to them.

In my studies, it was a lot of work to not only do research but also develop a research tool that is mature enough to be released on the Google Play store and that gets good reviews. Actually too much work for one person. Therefore I would like to thank all the students who helped me to develop FlexiSketch: Patrick Aeschbacher, Muhamed Ahmetovic, Sven Brunner, Stefan Bublitz, Markus Cadonau, Rita Forster, Fabian Gautschi, Sebastian Golaszewski, Thavorith Hean, Sönke Klüss, Oliver Lang, Catrin Loch, Manuel Martin, Milena Perelygina, Yulia Schmitt, Samuel von Stachelski, Tim Sterchi, Dario Todaro, Sacha Uhlmann, Marcel Weber, Kevin Wieser, Daniel Würsch, Gabriel Zimmerli, and Mengia Zollinger.

Last but not least, I want to thank my family and friends. They are always very supportive and cover my back, although some of them do not even know exactly what it means to do a PhD. Nonetheless, they always helped me in every way they could.

Abstract

Most software modeling tools only support predefined modeling languages. They are not flexible enough for creative requirements elicitation and early design sessions because their restricted vocabularies cannot cope with the diversity of emerging ideas. Therefore, software and requirements engineers frequently use physical media such as whiteboards and paper to sketch models and communicate ideas. The resulting model sketches, or photographs thereof, are not amenable for processing by software modeling tools because they are represented as image files. As such, they come without explicitly defined model syntax and semantics. Instead, modelers either ignore the sketches or extract important information from them later in the software engineering process, using this information to manually build semi-formal models from scratch. This manual re-creation can be time-consuming and error-prone, because sketches are often ambiguous. It is difficult to disambiguate them since contextual data is usually not stored within the sketches, and the original intentions behind them might no longer be known.

In this thesis, we propose a new, flexible modeling approach that enables a more seamless, semi-automatic transition from sketches to models. Our tool-supported approach combines sketching with lightweight metamodeling in a single modeling environment. We support free-form sketching similar to whiteboards and paper. In addition, sketched elements are recognized as individual constructs. A simple modeling language can be defined by annotating the constructs with types and constraints. The software automatically infers a metamodel by analyzing the whole sketch and the user's annotations. Thus, our approach supports several levels of formality. It enables the transformation of sketches into models, and paves the way for exporting them to other modeling or metamodeling tools. This thesis presents the conceptual solution of our approach, the FlexiSketch tool as a proof of concept and embodiment of this approach, and an evaluation of the approach in the form of initial studies with students and practitioners. The study participants deemed our approach to be valuable and pointed out that the simplicity and flexibility of our tool make it much more suited for creative work than other software modeling tools. Overall, the studies show that our approach is a successful example of blending advantages of whiteboards and software modeling tools, in order to obtain a flexible modeling tool that enables a powerful integration of sketches into the software engineering process. Furthermore, the studies confirm an industry need for such flexible tools.

Zusammenfassung

Die meisten Software-Modellierungswerkzeuge unterstützen nur vordefinierte Modellierungssprachen. Sie sind nicht flexibel genug für die kreative Anforderungsermittlung und frühe Design-Phasen, weil ihre eingeschränkten Vokabulare die Diversität entstehender Ideen nicht bewältigen können. Deshalb benutzen Softwareingenieure und Anforderungsanalytiker regelmässig physische Medien wie Whiteboards und Papier, um Modelle zu skizzieren und Ideen zu kommunizieren. Die resultierenden Skizzen, oder Fotografien davon, sind nicht zugänglich für eine Weiterverarbeitung durch digitale Modellierungswerkzeuge, weil sie als Bilddateien gespeichert sind. Als solche kommen sie ohne explizit definierte Modellsyntax oder Semantik daher. Stattdessen ignorieren Modellierer entweder die Skizzen im weiteren Verlauf des Softwareentwicklungsprozesses, oder sie extrahieren wichtige Informationen aus den Skizzen, und nutzen sie, um manuell von Grund auf neue teilformale Modelle zu erstellen. Dieses manuelle Kopieren von Information kann zeitaufwändig und fehleranfällig sein, da Skizzen oft mehrdeutig sind. Es ist schwierig, diese Mehrdeutigkeiten aufzulösen, da Kontextinformationen üblicherweise nicht zusammen mit Skizzen gespeichert

werden, und die ursprünglichen Absichten sowie Ideen hinter den Skizzen inzwischen möglicherweise nicht mehr bekannt sind.

In dieser Dissertation präsentieren wir einen neuen, flexiblen Modellierungsansatz, welcher einen nahtloseren und halbautomatischen Übergang von Skizzen zu Modellen ermöglicht. Unser werkzeugunterstützter Ansatz kombiniert Zeichnen mit leichtgewichtiger Metamodellierung in derselben Werkzeugumgebung. Wir erlauben freies Zeichnen genauso wie dies Whiteboards und Papier tun. Zusätzlich werden skizzierte Elemente als individuelle Konstrukte erkannt. Eine simple Modellierungssprache kann definiert werden, indem man die Konstrukte mit Typen und Einschränkungen annotiert. Die Software leitet automatisch ein Metamodell ab, indem sie die ganze Zeichnung sowie die Benutzer-Annotationen analysiert. Somit unterstützt unser Ansatz mehrere Formalisierungsgrade. Er ermöglicht die Transformation von Skizzen zu Modellen und ebnet den Weg für deren Export zu anderen Modellierungs- und Metamodellierungs-Werkzeugen. Diese Dissertation präsentiert die konzeptionelle Lösung unseres Ansatzes, das FlexiSketch-Werkzeug als Machbarkeitsnachweis und Manifestation des Ansatzes, sowie eine Auswertung des Ansatzes in der Form von initialen Studien mit Studenten und Fachkräften aus der Industrie. Die Studienteilnehmer haben unseren Ansatz als wertvoll erachtet und betont, dass unser Werkzeug aufgrund seiner Einfachheit und Flexibilität viel besser für kreative Arbeiten geeignet ist als andere Software-Modellierungswerkzeuge. Insgesamt zeigen die Studien, dass unser Ansatz ein erfolgreiches Beispiel dafür ist, wie man Vorteile von Whiteboards und Software-Modellierungswerkzeugen vereinen kann, um ein flexibles Modellierungswerkzeug zu erhalten, welches

eine mächtige Integration von Skizzen in den Softwareentwicklungsprozess ermöglicht. Die Studien bestätigen ausserdem, dass in der Industrie ein entsprechender Bedarf an solchen flexiblen Modellierungswerkzeugen besteht.

Contents

Abstract **vii**

Zusammenfassung **ix**

1 Synopsis **1**
1.1 Background and State of the Art 4
1.2 Motivation 19
1.3 Research Goal and Questions 23
1.4 Overview of Our Flexible Modeling Approach . . . 31
1.5 Contributions 42
1.6 Research Methodology 42
1.7 Chapter Summary 44

2 FlexiSketch: A Mobile Sketching Tool for Software Modeling **51**
2.1 Introduction 53
2.2 Flexible Sketch-Based Modeling 55
2.3 The FlexiSketch Prototype 59
2.4 Evaluation 67

2.5 Experiment #1: Feasibility of our Approach and Tool Usability 68
2.6 Experiment #2: Utility of the Approach 76
2.7 Threats to Validity 82
2.8 Related Work 83
2.9 Conclusion and Future Work 86

3 FlexiSketch TEAM: Collaborative Sketching and Notation Creation on the Fly 91
3.1 Introduction 93
3.2 FlexiSketch Basic 95
3.3 FlexiSketch Team 98
3.4 Preliminary Evaluation Results 101
3.5 Related Work 104
3.6 Conclusion 105

4 Semi-Automatic Generation of Metamodels from Model Sketches 107
4.1 Introduction 109
4.2 Modeling Languages and FlexiSketch 111
4.3 Metamodeling in FlexiSketch 114
4.4 Initial Evaluation Results 125
4.5 Related Work 127
4.6 Conclusions and Future Work 129

5 An Investigation of Participant Behavior in Computer-Supported Collaborative Notation Creation and Sketching 131
5.1 Introduction 133

5.2 FlexiSketch 135
5.3 FlexiSketch Team 137
5.4 Study Goal and Method 142
5.5 Analysis 146
5.6 Results 148
5.7 Discussion of Results and Design Implications . . . 162
5.8 Threats to Validity 166
5.9 Related Work 167
5.10 Conclusions and Future Work 169

6 Lightweight End-User Metamodeling with FlexiSketch 171
6.1 Introduction 173
6.2 A Tool-Supported Approach for Flexible Modeling 178
6.3 Study 1: What patterns emerge when modelers collaboratively define modeling languages? 189
6.4 Study 2: Can modelers define lightweight modeling languages correctly and completely? 209
6.5 Discussion 229
6.6 Related Work 246
6.7 Conclusions 251

7 Conclusion 253
7.1 Thesis Summary and Contribution 253
7.2 Revisiting the Research Questions 255
7.3 Next Steps 261

Bibliography 265

A Publications 289

A.1 Conference Papers 289
A.2 Journal articles 290
A.3 PhD Symposium 291
A.4 Technical Reports 291

List of Tables

2.1 The 8 most frequently mentioned feature wishes. . 73
2.2 Diagrams drawn on whiteboards and paper as reported by participants. 78

5.1 Student answers regarding FlexiSketch features on a Likert scale from "strongly disagree" to "strongly agree". 153
5.2 The amount of symbols, links, and defined types contained in each diagram from students (left) and practitioners (right). 157
5.3 A summary of the results, grouped by research question. 162

6.1 The amount of symbols, links, and defined types contained in each diagram from students (left) and practitioners (right). 196

6.2 Practitioner demographics: their work field (SE/RE), the years of work experience, experience with touch devices ((H)igh/(M)edium/(L)ow), and metamodeling knowledge (Yes/No). 210
6.3 Amount of data points received. 213
6.4 Completeness and correctness of student models and metamodels (without cardinality rules). 218
6.5 Statistical significance of the differences between paper and FlexiSketch results. 218
6.6 Comparison of times between students working with paper and FlexiSketch. 226
6.7 Completeness (%) of (p)ractitioners' metamodels before (v1) and after (v2) consulting the wizard. . 227

List of Figures

1.1 Current tool support is divided into support for informal modeling (sketching) and (semi-)formal modeling. 10
1.2 We intend to bridge the gap between informal and formal modeling tools with a tool that combines the flexibility of informal tools with formalization abilities. 21
1.3 Overview of the research questions. 26
1.4 The FlexiSketch tools (with the Android version on the left, the desktop version on the right). 32
1.5 High-level scheme of our approach. 33
1.6 Overview of our meta-metamodel in textual form and as UML class diagram. 38
1.7 A model and its underlying custom modeling language were exported from FlexiSketch to MetaEdit+. 40
1.8 A UML activity diagram shows how our approach can be used. 41

1.9 Thesis roadmap: the mapping between papers and research questions. 45
1.10 The main focus of each chapter. 49

2.1 The three activities leading to a semi-formal model in the end. 57
2.2 Screenshot of the FlexiSketch prototype. 60
2.3 Sketch recognition beautified a hand drawn use case diagram. 66
2.4 The 5 most frequently occurred usability problems. 71
2.5 Most frequently stated reasons why paper or whiteboards are used. 79

3.1 The Android version of FlexiSketch running on a tablet, and the desktop version running on a Mac (showing two different model sketches). 96
3.2 Metamodel excerpt from the tablet sketch in Figure 3.1. 97
3.3 Meeting participants collaboratively create and discuss a model. The workspace is synchronized across the tablets and the electronic whiteboard. 99
3.4 A symbol on the left tablet is selected and appears in blue. On all other tablets, the symbol appears in red and is locked. 101
3.5 Result extracts from practitioner teams. The grey boxes show defined elements. Here, types are also displayed to the bottom right of each element. . . . 103

4.1 The FlexiSketch tool showing a user's sketch. . . . 113

4.2 Inferring example. 120
4.3 The wizard highlights an instance of a connection type and asks for the cardinalities. 122
4.4 A close-up of the wizard window. 123
4.5 A minimalistic class diagram fragment. 125
4.6 Metamodel of the fully defined class diagram fragment from Figure 4.5. 126

5.1 Screenshot of FlexiSketch showing the UI and a model sketch. 135
5.2 Meeting participants collaboratively create and discuss a model sketch using multiple tablets and a synchronized drawing canvas. 140
5.3 Left tablet: a symbol is selected and appears in blue. Right tablet: the symbol is locked and appears in red. 141
5.4 Phases of editing and discussions in student groups. 149
5.5 Phases of editing and discussions in practitioner groups. 150
5.6 Extracts from the results of practitioner groups, a) G4, b) G5, c) G6. The grey boxes show the defined elements. 155
5.7 Talk category distribution in student and practitioner groups. 156
5.8 When participants talked about new types – before, during, or after defining them. 158

6.1 Screenshots of the mobile and desktop versions of FlexiSketch showing the UIs and some model sketches. 179
6.2 The cardinality dialog. 182

6.3 A model sketch and its metamodel exported to MetaEdit+. 185
6.4 A group of practitioners is working in one of our simulated workshops. 192
6.5 Extracts from the results of practitioner groups, a) PG1, b) PG2, c) PG3. The grey bars show the defined types. 197
6.6 Phases of editing and discussions in student groups. 199
6.7 Phases of editing and discussions in practitioner groups. 200
6.8 Talk category distribution in student and practitioner groups. 205
6.9 When participants talked about new types – before, during, or after defining them. 206
6.10 Amount of metamodel information defined by students. 220
6.11 Amount of cardinalities defined by students. 224

Chapter 1
Synopsis

Models play a central role in software engineering. A model can be defined as "*an abstract representation of an entity for a given purpose*" [Jea13] (based on [RWLN89]). Most artifacts produced during software development can be seen as models of the software system to be built: requirements described in textual and graphical form, diagrams depicting concepts and architectural solutions, and even source code. The latter is a textual model that describes the software system and is used by a compiler to create an executable program. France and Rumpe [FR07] conclude that software development is, for the most part, a model-based problem solving activity.

A paramount advantage of models is that they allow for abstraction. The reduction criterion is one of the three main model characteristics defined by Stachowiak [Sta73], and states that a model does not include all attributes of an original, but only those that are of interest to the modeler. This means that models can be built at

different levels of abstraction, and they can provide different views on the same subject. As such, they are an important tool during the whole software engineering process for depicting problems and solutions in abstract and lucid ways, and facilitate – among other things – the reasoning about them [Hut95].

In this work, we focus on the use of semi-formal graphical models in requirements elicitation and early design activities, when it is not yet clear what the software system solution will look like. In these situations, creativity is a significant part of the job. Models help to get an overview of the problems at hand and can be used to draft requirements and design ideas. Models created in this context have two interesting characteristics. First, they often exist as sketches. Sketching fosters creativity [Goe95, GD96], because it gives complete freedom in what engineers can draw. Engineers can focus on idea generation and do not have to create nice-looking models. Second, the models are often of an informal or semi-formal nature and do not adhere to standard modeling languages [CVDK07, DH07]. Engineers choose whatever notations suit them best for depicting their ideas. This helps i) to keep their cognitive effort low, and ii) to communicate their ideas to others in meetings. The latter is not a trivial task, because software projects have many different stakeholders. According to Glinz and Wieringa [GW07], a stakeholder is defined as "*a person or organization who influences a system's requirements or who is impacted by that system*". Stakeholders come from different areas of expertise, and many of them do not have a background in computing or engineering. They might not understand specific modeling languages such as UML. Thus, it makes sense to choose

simple, ad-hoc notations depending on the modeling knowledge of the other stakeholders attending the meeting. Furthermore, creative sessions should not be hampered by the task of mapping the ideas of modelers to specific languages, which disrupts the creative flow. To avoid this, modelers deliberately deviate from standard notations as needed. Not being constrained to specific modeling languages fosters creativity in early project stages [DH07].

Today, engineers create model drafts on physical media such as whiteboards and paper, and later have to manually re-create the models in software modeling tools if they want to re-use them. With the term *software modeling tools* we mean software that allows its users to model requirements and designs of software systems. A typical example are tools for creating UML models.

We think that engineers would benefit from a solution that allows them to refine their informal model sketches into semi-formal models instead of re-creating models from scratch. Indeed, researchers argue that there is a need for more flexible modeling tools (see, for example, [CVDK07, OJDB10, WHD+11]). Developing a tool-supported approach for flexible modeling that allows its users to formalize model sketches is the focus of this thesis. The state of the art only supports a refinement of model sketches for specific modeling languages and notations. As soon as engineers take the freedom to choose arbitrary modeling languages, the formalization of sketches becomes a manual process due to a lack of tool-support.

1.1 Background and State of the Art

In this section, we first briefly define the terms *sketch*, *model*, *informal*, and *semi-formal*. We then give an overview of the state of the art regarding flexible modeling approaches. In particular, we show that, although the need for a combination of sketch-based interfaces with formal modeling was recognized long ago [Sut63], and there exist software modeling tools that support sketching, today there is still a gap between sketches and models that is not bridged by modeling tools. We discuss the trade-off between flexibility and formality that is responsible for the gap, and discuss the inevitable use of metamodeling to overcome this trade-off.

1.1.1 About Sketches, Models, and Formality

In the remainder of this thesis, we will repeatedly use the terms *sketch* and *model*, and we will speak about transforming sketches into models. We will also distinguish between informal and (semi-)formal artifacts. Therefore, in this section we provide definitions of these terms.

Sketches versus models. Stachowiak [Sta73] defined a scientific model as a representation of an existing entity or an entity to be built. With this categorization of artifacts into models and non-models, we will not be able to sufficiently distinguish between the terms *model* and *sketch*, because most sketches will also be

classified as models. For example, someone could draw a picture of a computer, and this picture would be both a sketch and a model. Instead, we use a more restrictive definition for the term *model* and expand the definition from Stachowiak. We categorize a drawing as a model depending on the formality level of the drawing.

Definition: Sketch. A sketch is a rough drawing.

Definition: Model. A model is a representation of an existing entity or an entity to be built. It consists of multiple components and adheres to a set of syntactical rules that describe how the components can be related to each other. The components have some intended semantics.

The model rules and semantics can exist explicitly as part of a modeling language specification, or implicitly in the mind of the modeler. While a model implies the existence of some syntactic and semantic rules, a sketch can be any type of informal drawing, for example a painting, but it can also denote a drawing of a model. Therefore, *model* and *sketch* are not disjunct categories. The term *model* implies some level of formality, while the term *sketch* characterizes how something is drawn (i.e., a model can be drawn in a fast and rough way, in which case the model is also a sketch).

Brambilla et al. [BCW12] distinguish between sketching and modeling activities in a similar way. They say that, when modeling is performed, the created artifacts "*have implicit but unequivocally defined semantics which allow for precise information exchange and many additional usages. Modeling, as opposed to simply drawing,*

grants a huge set of additional advantages, including: syntactical validation, model checking, model simulation, model transformations, model execution [...]".

Sometimes we use the terms *model sketch* or *sketched model*. We do this to make clear that, although the sketch depicts a model, a software tool cannot recognize it as such, but only sees it as an image. The important difference is that humans see the drawing as a model (they have implicit syntactical rules and semantics in mind), but a software tool does not (it is not aware of any syntax or semantics).

Definition: Model sketch / Sketched model. A rough drawing of a model without explicitly defined syntax and semantics.

When we talk about *transforming sketches into models* in this thesis, we mean the process of enriching a sketch with explicit syntax and/or semantics that previously only exist in an implicit form.

In this thesis, we focus on semi-formal graphical models consisting of nodes and edges (i.e., graphs), and we use the term *diagram* synonymously. This, for example, includes UML models, context diagrams, and diagrams consisting of generic, undefined nodes and edges. When we mention the term *models* in the remainder of this thesis, we mean this type of graphical node-and-edge model, unless stated otherwise. Besides this type of model, there exists a wide range of structures, sketches, and models, where entities are not divided into nodes and connecting edges. Examples are tables, lists, hierarchies where the connections between elements

are not visualized with distinct graphical elements (e.g., when a symbol encompasses other symbols), various model types where spatial relationships (the relative locations of symbols to each other) convey meaning (e.g., the Jackson diagram that shows executed functions in chronological order from left to right), architectural diagrams, and construction plans. It is not possible to consider such a wide variety of structures found in sketches and models in the scope of this thesis. However, the node-and-edge diagram is the most frequently used model type in the context of early requirements elicitation activities, as we will show in Chapter 2. Therefore, it makes sense to focus on this type of model.

Informal vs. (semi-)formal. We often mention the terms *informal* and *(semi-)formal* to denote the amount of restriction modelers adhere to when they are sketching.

Definition: Informal. When modelers use free-form sketching and do not think about any particular modeling language, we say that they draw an *informal* sketch.

Definition: (Semi-)formal. We use the term *(semi-)formal* when (some of) the elements of a drawing have a meaning, and when there are syntactical rules that state how different elements of the drawing can be connected to each other.

It does not matter whether the syntactical rules exist explicitly, or only implicitly in the mind of the modeler (the modelers can also be unconsciously mixing different modeling languages). There is no strict separation of *informal* and *semi-formal*; it is rather a smooth

transition. Also, since syntax and semantics may exist implicitly in the modeler's mind, it is difficult to objectively categorize a drawing. However, when we take the viewpoint of a software tool, the distinction becomes easier: for a software tool to see a drawing as a semi-formal model, some explicitly defined, machine-readable syntax and/or semantics need to exist. Otherwise, the software tool sees a drawing as an informal sketch. Therefore, from the viewpoint of a software tool, it can be said that a drawing is either a sketch and completely informal, or a model and at least partly formal (semi-formal).

1.1.2 Stuck on Predefined Levels of Formality

The idea to provide sketch-based interfaces together with formalization capabilities is not new. Researchers recognized early the need for an input option that feels natural. Sutherland presented Sketchpad back in 1963, using a digital pen for drawing input [Sut63].

In 1996, Gross and Do published a paper about the Electronic Cocktail Napkin [GD96]. As the name implies, it lets its users create the same kind of free-form sketches and drafts that they would draw on napkins when they have an idea but no paper is readily available. Their approach allows for user-defined glyphs and combinations of glyphs that can then be identified by a sketch recognition algorithm. Their focus does not lie on the formalization of sketched constructs, but on sketch refinement with the help of sketch recognition and other features such as providing different

abstraction levels and beautified representations. Glyphs can also further be defined by adding constraints on how they can be positioned and scaled on the sketch canvas.

Despite this early effort, today's software modeling tools support only predefined levels of formality. In contrast, software engineering (SE) processes are often iterative processes where artifacts pass through many levels of formality: ideas evolve into sketches, which in turn evolve into models. Current approaches for sketching and modeling do not consider this evolution enough [OJDB10]. They lead to tools that support modeling only at one specific level of formality. We argue that a flexible modeling tool is only then truly flexible when it supports many levels of formality, and equally important, the transitions between these levels.

1.1.3 A Compromise Between Freedom and Formalism

Developing a truly flexible modeling tool comes with a big challenge. This challenge is the trade-off between freedom and formalism. Freedom is maximized when a tool supports free-form drawing. However, as soon as it becomes desirable to introduce some kind of formalism, the degree of freedom decreases because the modeler needs to adhere to certain modeling language rules in order to reach the desired formalism. Many existing approaches have avoided the trade-off by only supporting either the informal side or the formal side of software and requirements modeling (Figure 1.1).

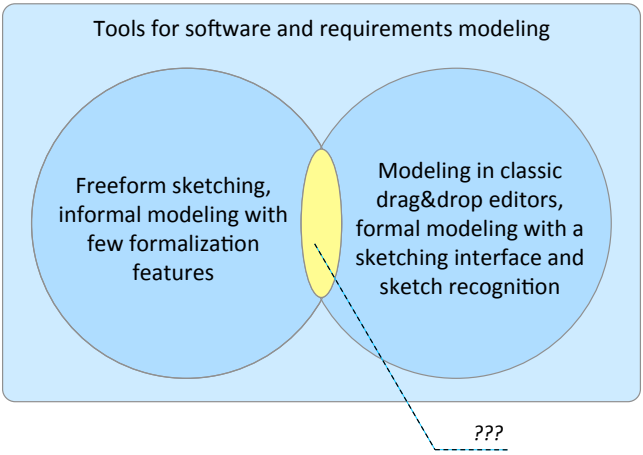


Figure 1.1: Current tool support is divided into support for informal modeling (sketching) and (semi-)formal modeling.

In other words, many software tools are either a drawing tool or a modeling tool [BCW12]. If we create an artifact with a drawing tool, it will always be a sketch from a technical perspective, because the tool/the computer cannot interpret the drawings. If we want to create a machine-processable model, we have to use a modeling tool instead. There exist few tools that are both a drawing tool and a modeling tool. Some examples are InkKit [PF07] and Scribble [SA13], where drawn strokes can be recognized and translated into model elements. Another interesting example is the BITKit prototype from Ossher et al. [OBS+10], as part of their work for blending the advantages of office tools and modeling tools. The mentioned tool examples combine drawing and modeling capabilities: they do not just allow users to perform drawing and

modeling side by side, but drawing entities can be transformed into model entities. Users can create entities that are just drawings at first and do not convey a machine-readable meaning, and then they can convert these entities to model elements later on.

Researchers are aware of the importance of sketching for design, creativity and idea generation [Goe95, CC09, WSN+11]. Sketching fosters discussions and trying out different ideas and alternatives. Therefore, one idea is to take (semi-)formal modeling tools and augment them with sketch interfaces. Sketch recognition algorithms compare drawn symbols with constructs from the supported modeling languages and notations. Examples of this idea are found in tools such as Tahuti [HD06], SUMLOW [CGH08], and Knight [DHT00]. The sketch interface cannot change the fact that these tools only support certain predefined modeling languages. SUMLOW and Knight provide guidance to formalize sketched constructs that do not comply with the supported modeling languages. But this means that the sketched constructs have to be altered until they conform to the given languages. Thus, such tools do not increase the type of flexibility that we are looking for.

Other tools such as SketchREAD [AD04] and InkKit [PF07] allow for the integration of additional modeling languages, either by programming or library plug-ins. These mechanisms are beneficial if a company wants to introduce a custom modeling language that is then used on a regular basis. But requirements engineers are not programmers, and it is too much effort to programmatically create new modeling languages in order to achieve a formalization of sketches. In that case, it would likely be easier to manually

translate the information using an existing language such as UML. In conclusion, augmenting formal tools with sketch interfaces does not increase their flexibility.

On the other side, there are approaches focusing on the support of informal sketching as part of early design and requirements elicitation phases. Examples include Calico [MBD+10] and Sketch for Eclipse [SB10]. The formalization features of such tools are limited and do not allow the creation of custom modeling languages.

Regarding the tradeoff between freedom and formalism, the question is whether it is possible at all to develop a tool that can be truly categorized as belonging into the middle of Figure 1.1. In order to achieve this, it is not enough to just provide an very large collection of modeling language libraries. Still, users must have the option of somehow translating their sketched constructs into formal ones. One possible solution to this dilemma is end-user metamodeling. With end-users, we mean users of modeling languages and modeling tools who have no or little metamodeling knowledge. One way of approaching end-user metamodeling is described by Ossher et al. [OBS+10]: the tagging mechanism of their BITKit prototype allows users to tag constructed entities with multiple (arbitrary) terms, which leads to a classification of the entities. The theoretical framework of Ossher et al. also introduces the notion of structure definitions for the description and re-use of structures, but this idea is not yet found in their tool prototype.

1.1.4 Metamodeling

Multiple parties can only interpret and reach a common understanding of a model if they share the same metamodel. A metamodel describes the model and its parts – be it implicitly in persons' minds, or explicitly as a machine-readable file consisting of parsable definitions and rules. The same is true for refining sketches into models that are to be interpreted by a software modeling tool. Therefore, some form of metamodeling is needed if users are allowed to come up with arbitrary modeling languages. Without a metamodel, a software tool can at best recognize the individual components of a model as distinct parts. The tool needs a metamodel and a semantics description of the individual components in order to know what these components actually mean. While an explicit semantics description is not within the scope of this thesis, a metamodel is needed if sketches are to be transformed in anything that goes beyond general diagrams consisting of undefined nodes and edges.

Definition: Metamodeling. The process of creating a modeling language and notation. A modeling language is defined by a concrete syntax (the notation), an abstract syntax (the individual elements of the language and rules how these elements can be related to each other), and the semantics (the meaning of the language constructs) [AK03, Kle08].

Definition: Metamodel. A metamodel is the abstract syntax represented as a model (often in the form of a UML class diagram).

The concrete syntax of a modeling language is usually not regarded as being part of the metamodel itself [AK03, Kle08], but is a distinct part of the modeling language description.

Finally, when users can create metamodels, the language in which these metamodels can be stored – a meta-metamodel – must also be explicitly defined. In the same way as two different software tools need to share the same metamodel in order to allow for an import/export of models between them, two tools need to share the same meta-metamodel in order to allow for an exchange of metamodels between them.

Definition: Meta-metamodel. A meta-metamodel describes the components and the syntactical rules for a metamodel (in the same way as a metamodel describes the components and rules for models).

Metamodeling tools are usually tailored for being used by language designers or engineers with object-oriented programming knowledge and not easy to use. MetaEdit+ is one of the most popular metamodeling tools that are commercially available [KLR96]. It provides a suite of metamodeling editors for the creation of modeling languages. Once a language is ready, the user can push a button which generates a modeling editor that conforms to the created modeling language. Metamodeling and sketching/modeling is never done in the same tool, and thus users must decide which of these two activities they want to perform next, and switch to the corresponding tool.

Some tools such as MaramaSketch [GH07] and the Electronic Cocktail Napkin [GD96] provide a sketching interface. Apart from

this, they have similar advantages and disadvantages as other metamodeling tools: the target users are metamodeling experts, and the tools are too heavyweight in order to be used in early, creative design and requirements elicitation activities.

1.1.5 Lightweight End-User Metamodeling

Metamodeling is a complex topic and not easy to do in the right way [QGW10, SCDLG12]. A lot of literature can be found about principles and design guidelines for modeling language design. For example, Paige et al. [POB00] name nine principles for language design. The principles could also be regarded as quality attributes to judge the quality of a designed modeling language. Karsai et al. [KKP+09] present 26 design guidelines for domain specific languages. It is believed that only metamodeling experts, also called language designers, can create high quality modeling languages [BCW12, Kle08]. Therefore it is not surprising that, historically, the creation and use of a custom modeling language is always a linear process: first, the language designer creates a modeling language, and afterwards it is used by engineers to create models [Kle08]. We agree that creating good modeling languages is difficult and cannot be done without experts in language design. However, we argue that the users of a modeling language (e.g., requirements and software engineers) need to be better involved in the language design process, which should consist of multiple iterations. This has been recognized, especially with the advent of model driven engineering (MDE). For example, some of the design

guidelines stated in [KKP+09] make clear that it is important to involve language users in the creation process, i.e., a good language design process also consists of a requirements engineering part. Therefore, researchers tried various approaches to make metamodeling accessible for end-users. The main focus of this works is usually to come up with a high-quality metamodel, and thus the metamodeling features they provide become too complex for non-expert language designers.

Many metamodeling tools (also called meta-CASE tools), such as Metaview [Fin94] and MetaBuilder [FHH00], use a programming language for defining some parts of a modeling language. Quattos et al. performed a study in which they compare different, more user-friendly alternatives for defining model constraints (e.g., cardinality rules) [QGW10]. Especially, they compare a wizard approach with constraint definition by example, and found that definition by example resulted in a higher number of correct user definitions.

Cuadrado et al. propose a similar approach to definition by example, called bottom-up metamodeling, and emphasize the importance of having an iterative language creation process [SCDLG12]. Domain experts can create example models in a tool with sketching facilities. The language designer imports these models into a specialised tool, annotates the model fragments, and lets the tool create the metamodel. In this approach, different tools are used for modeling and metamodeling.

Cho et al. present a framework that assists in creating a modeling language [CGS12]. The semi-automated approach takes as

input a set of user-created models, and infers as much metamodel information as possible. It presents the results to the language designer who can then change and adjust the metamodel as needed. Similarly, Javed et al. have developed Mars, a tool to re-construct a metamodel for a set of models created with the same language [JMGB08]. Their goal is to infer a metamodel in cases where the original metamodel or the language description is lost. While we can borrow some of the ideas for metamodel inference, our situation is different because sketching in early phases of requirements elicitation and design often leads to individual models, rather than a set of models conforming to the same modeling language. Less example data means that we cannot automate metamodel creation to the same extent as the related approaches.

Cánovas Izquierdo et al. propose an approach to better engage end-users in the process of modeling language development [ICLF+13]. They present an enhancement of the Eclipse plugin Collaboro, which allows to collaboratively develop domain specific modeling languages by example. However, the approach does not use a sketch-based interface. Furthermore, an Eclipse plugin is not the best tool choice for creative requirements elicitation and design sessions. Cánovas Izquierdo et al. focus on an iterative refinement of the metamodel rather than the creation of an initial metamodel for formalizing sketches.

Ossher et al. [OBS+10] propose an approach to blend the advantages of office tools and modeling tools. Office tools usually provide a set of predefined shapes and styles, but do not focus on sketching. However, the core of their approach to end-user

metamodeling can be equally applied to a sketching environment: users perform metamodeling by example by tagging entities on the drawing canvas with multiple terms in order to classify the entities. In addition, structure definitions are used to create syntactical rules. Ossher et al. present a comprehensive theoretical framework. However, not all of their ideas are incorporated into their BITKit prototype. For example, tagging is possible, but structure definitions cannot be created by users. Also, it is unclear how BITKit could support the export of a metamodel, or the modification of a model such that it conforms to an existing modeling language and can be exported to another modeling tool.

There are two more important topics related to metamodel creation. The first is managing the co-evolution of models and metamodels, and is addressed in, e.g., [GSFV14, CGS12, ICLF+13, CREP08]. The second topic is the assessment of the quality of the created artifacts (i.e., the models and metamodels), and is studied in, e.g., [SNH+09, POB00, Jea13]. These two topics are beyond the scope of this thesis, but are pointers for possible future work.

To sum up, the work presented in this section provides some ideas and methods for end-user metamodeling, but it does not deliver a satisfying solution that could be incorporated into a sketch-based software modeling tool to be used in early, creative requirements elicitation and design phases. The approaches presented here focus on metamodeling as the main task of the end-users, while we search for solutions for creative, sketch-based modeling where metamodeling exists in a lightweight form and is as invisible as possible to the end-users.

1.2 Motivation

Using natural language to document software requirements and conceptual solutions can often be imprecise and error-prone [Rup14]. For the modeling of requirements and software, diagrammic notations have provided a great deal of help as alternative to and complement for natural language specifications. There is a wide variety of graphical modeling languages that can be used. Different models support different levels of abstraction and have the ability to convey information more precisely and with less misunderstandings than natural language. Some of the models found in software projects are very detailed and of a very formal nature, e.g., models that can be verified and validated against a requirements specification, models that allow for some sort of simulation, and models that can be automatically translated into source code. Other models are informal and less detailed, e.g., model drafts, models that convey new ideas, models that are created to discuss certain aspects of a problem or solution in order to reach a common understanding, or models that give an overview of a problem or system.

1.2.1 A Gap in the Modeling Tools Landscape

As far as tool support for software and requirements modeling is concerned, engineers can choose among many different tools. They

can use tools such as the Eclipse Modeling Framework[1] or Enterprise Architect[2] for formal modeling, use more general purpose modeling editors and office tools such as Microsoft Powerpoint and Visio for creating informal models, or use pure sketching applications as well as physical media such as paper and whiteboards for completely free-form sketching.

At a first glance, it seems that tool support for requirements and software modeling is comprehensive. However, as discussed in Section 1.1.3, there is a part of the spectrum that is not supported by the state of the art. While many tools support the user in semi-formal and formal modeling, and pure sketching applications allow for free-form sketching, there is a lack of flexible software modeling tools that support creative requirements elicitation and ideation activities [OBS+10, OJDB10]. Figure 1.2 depicts this issue in an abstract way. Informal modeling tools are flexible in the sense that they allow for any kind of drawings, but they lack formalization capabilities. Formal modeling tools allow for formal and precise modeling, but they lack the flexibility of informal tools. A gap between these two types of tools embodies the incompatibility between sketches and more formal models.

This thesis aims at bridging the gap with a new approach for flexible modeling. The approach is supposed to enable a semi-automated formalization of model sketches. In a perfect world, the approach would retain the full flexibility of informal tools, i.e., the cross in Figure 1.2 would be on the same height as the informal

[1] https://eclipse.org/modeling/emf/ [last checked: 12/05/15]
[2] http://www.sparxsystems.com.au/ [last checked: 12/05/15]

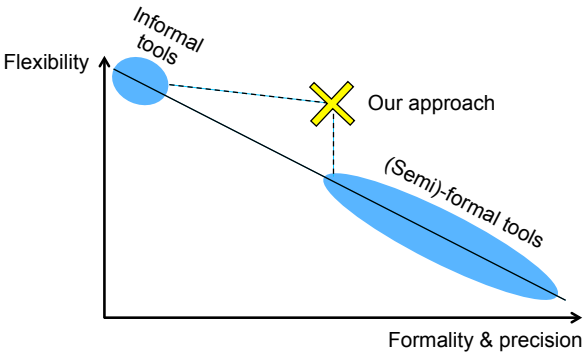


Figure 1.2: We intend to bridge the gap between informal and formal modeling tools with a tool that combines the flexibility of informal tools with formalization abilities.

tools. However, this would require a lot of end-user metamodeling, while one of our concerns is to lower the amount of needed end-user metamodeling to a level that is acceptable for modelers who want to perform modeling in the first place and do not want to spend much time on metamodeling.

1.2.2 The Media Disruption Between Sketches and Models

The motivation of this thesis is to avoid the *media disruption* between sketches and models. This disruption happens when engineers create model sketches on physical media and then want to digitize and re-use the sketches in software modeling tools.

Definition: media disruption. The action of manually changing either i) the medium on which information is stored, or ii) the form in which the information is stored, in order to make the information amenable for further automated processing. This action has a negative impact on the overall process time and cost because a manual translation (compared to an automatic one) is usually time-consuming and error-prone.

A model sketch created on physical media can be preserved and digitized by taking a photograph, a model sketch created with a painting application can be stored as an image file. In both cases, the result is a single image file that contains the whole model sketch as a single entity – the model sketch cannot be analyzed and processed by a software modeling tool. As a consequence, engineers need to manually re-create the information conveyed in a sketch as semi-formal or formal model if the information needs to be processed further in the SE process. The media disruption when manually re-creating sketched information is time-consuming, and, if it is not done directly after a sketch got created, it can be error-prone because some of the intentions behind the sketch (the context in which the sketch got created) might no longer be remembered.

Therefore, software and requirements engineers could benefit from flexible modeling tools when they perform creative activities such as brainstorming, idea generation, requirements elicitation, and the creation of early design. These activities happen mostly during early project phases, but also reappear at various points in time during projects due to the iterative nature of today's software

processes. As part of these activities, requirements engineers and other stakeholders often draw model sketches to write down and present their ideas. Semi-formal and formal modeling tools are too cumbersome to use in these moments because they hamper the creative flow: instead of just capturing ideas, engineers would also have to think about how to depict their ideas in the modeling languages that are supported by the respective tools. In addition, formal tools might require premature commitment to enter details and use formalisms that are overkill for the purpose of sketching models at different levels of abstraction [DH07]. In contrast, free sketching supports creativity and externalizing ideas [EP11, Goe95]. Therefore, engineers like to use physical media such as paper and whiteboards for creative tasks, because they allow for any kind of notation on any level of abstraction, and they are easy to use. But once engineers want to digitize and re-use the created artifacts, they are facing the aforementioned media disruption.

1.3 Research Goal and Questions

Until now, we have argued that new flexible modeling approaches can be beneficial for both requirements engineers (performing early requirements modeling) and software engineers (creating early software designs). Such an approach would need to cover a broad area of possible applications, and we do not intend to create a one-size-fits-all solution. For our research goal, we therefore narrow down the scope to requirements engineering (RE). We think that RE can benefit the most from such an approach because

the RE process includes many stakeholders who are unfamiliar with modeling languages used in software development, and thus non-standard, ad-hoc modeling languages are frequently used in RE.

The main goal of this thesis is to design a new tool-supported approach for flexible requirements modeling while overcoming the aforementioned problem of media disruption. The approach should allow for sketching requirements on a level as informal and flexible as possible, and at the same time support the formalization of these sketches such that they can be re-used and edited in other software modeling tools. With our approach, requirements engineers will have the choice of refining a sketch into a semi-formal model, as an alternative to how it is done today: creating a new model from scratch and manually copying and translating the information conveyed in a sketch to the new model. The approach should allow for enough formalization such that it can connect to semi-formal modeling approaches, thereby bridging the gap in Figure 1.2.

Thesis statement: *Informal model sketching can be combined with formalization mechanisms in a single tool, in order to enable requirements engineers to transform their sketches into semi-formal models and export them as such.*

As discussed in Section 1.1, a software tool can only interpret structures as semi-formal models and provide adequate editing support if it knows the metamodel according to which the models are built. Thus, the output generated from our approach has to contain metamodel information besides the sketched model.

One way of achieving this is to provide a large set of predefined modeling languages and requesting the user to only use provided language constructs while sketching. A very large language set might lead to the illusion that informal, flexible sketching is then possible. But in fact, it is not very flexible because users are restricted to the predefined language constructs – everything else cannot be interpreted and exported as a semi-formal model. By adhering to predefined language constructs, sketching is no longer informal. In order to reach the thesis goal, we therefore choose another method: we introduce a form of lightweight metamodeling as part of our approach, and give users the ability to define their own modeling constructs and languages. As a consequence, our research also includes questions about end-user metamodeling.

In order to deliver proof for our thesis statement, we define and answer the following four research questions in this thesis. Figure 1.3 shows how the questions are related to each other.

RQ 1: How do requirements engineers re-use information contained in informal model sketches? We started our work about flexible modeling with the hypothesis that requirements engineers use physical media for early requirements modeling and are then stuck with information that is cumbersome to edit and re-use. Therefore, we first asked an *exploratory question* to find out to what extent this problem really exists in practice: are physical media frequently used for informal modeling? What happens with the information contained in these models? Answering this question also showed us the needs of requirements engineers with regard to flexible modeling.

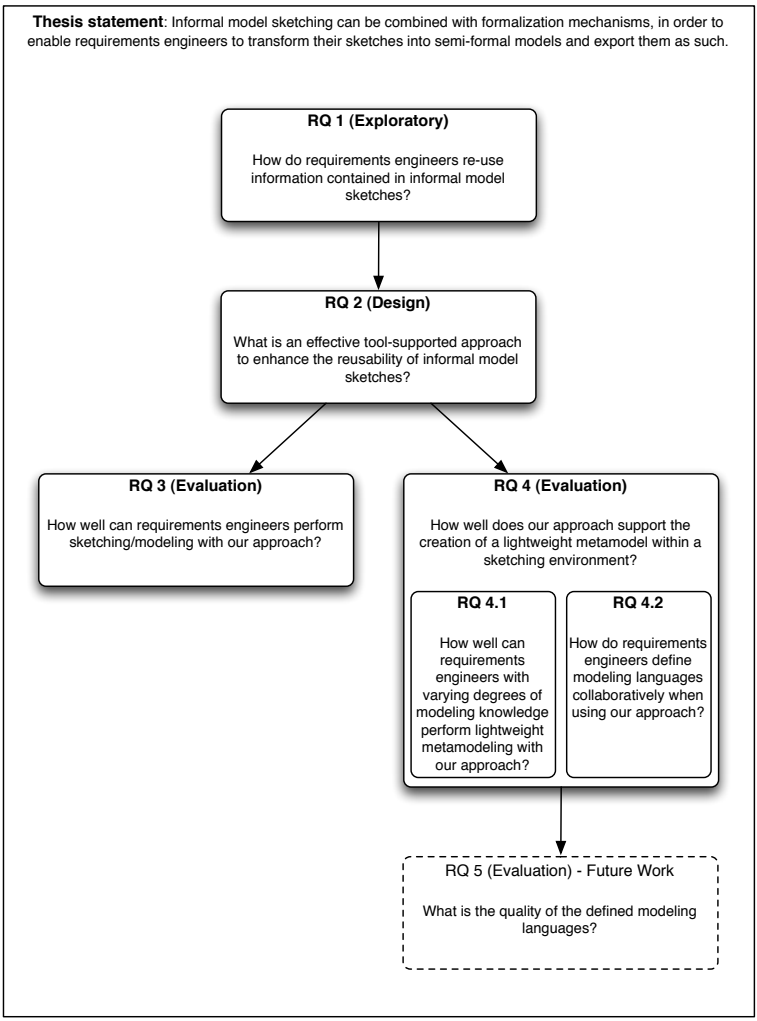


Figure 1.3: Overview of the research questions.

We addressed the question by a literature review and an exploratory study, in which we performed semi-structured interviews with requirements engineering practitioners and asked them about their daily work. Following this, we presented and discussed with them an initial tool prototype to find out whether our first ideas went into the right direction and to elaborate more on the engineers' needs.

RQ 2: What is an effective tool-supported approach to enhance the reusability of informal model sketches? This is a *design question* and builds on the answers from RQ 1. Our goal was to specify an approach and implement a software tool that is comparable to whiteboards and paper in terms of flexibility, but also satisfies the needs related to reusability.

We addressed the question by designing a conceptual solution according to the findings from RQ 1. The solution contains a free-form sketching interface and a form of lightweight metamodeling that allows users to define their sketched constructs. These definitions enable the export of sketched information as semi-formal models. We developed a tool prototype as proof of concept that we could use to evaluate our approach. The development followed an iterative process in which we enhanced the prototype according to feedback from test users.

RQ 3: How well can requirements engineers perform sketching/modeling with our approach? This *evaluation question* was the first step of evaluating our approach. We wanted to assess the usability of our tool in regard to sketching and informal modeling, and we wanted to know how much it satisfies the

informal modeling needs of requirements engineers. The sketching interface is the foundation of the tool, and the rest of our approach will be geared to it.

To address RQ 3, we performed a qualitative feasibility and utility study which included experienced requirements engineers as well as novice modelers (computer science students) working with our tool prototype. We asked practitioners to draw examples for diagram types that they would also draw on paper or whiteboards in their daily work, and we observed whether our solution provides adequate support for drawing and editing these diagram types.

RQ 4: How well does our approach support the creation of a lightweight metamodel within a sketching environment? This *evaluation question* investigates the effectiveness of our approach to make model sketches amenable for re-use.

As metamodeling is a big research topic on its own, we cannot cover it comprehensively, but we picked the parts that are of special interest in the context of this thesis, and divided RQ 4 into the following two subquestions:

RQ 4.1: How well can requirements engineers with varying degrees of modeling knowledge perform lightweight metamodeling with our approach? Related work shows that end-user metamodeling is hard to achieve (e.g., [QGW10]). Therefore, we wanted to find out whether requirements engineers can cope well enough with the lightweight metamodeling mechanisms of our approach such that our prototype receives all the information it needs to export model sketches to other modeling tools.

To address RQ 4.1, we performed a quantitative experiment with 107 students from two universities. The students had to draw a model according to a given problem, using a given modeling language. They also had to define all elements and cardinality rules on the metamodel level. We analyzed the results to see whether the students were able to do so. In order to identify how our particular tool prototype influenced the results, some students had to do the same assignment on paper. In order to compare the results with those of experienced modelers, we also conducted a similar, qualitative experiment with practitioners.

RQ 4.2: How do requirements engineers define modeling languages collaboratively when using our approach? To our knowledge, our tool is the first solution that allows multiple users to collaboratively define a modeling language in a sketching environment. This enables us to investigate the requirements engineers' collaborative metamodeling behavior and how they agree on a common notation. In contrast to working with paper or a whiteboard, the collaborators have to define their modeling constructs explicitly when using our approach (if they want to be

able to export their sketch as a semi-formal model). We wanted to know how this additional task affects the interaction patterns between the collaborators, with the results providing insights into how flexible tools could support lightweight end-user metamodeling in the future.

A qualitative study with groups of practitioners and groups of students addresses RQ 4.2. We investigated how small groups draw and define models collaboratively in early requirements elicitation sessions (i.e., sessions in project phases when many requirements are still unclear or unknown). Similar to RQ 3, we included both experienced modelers (practitioners) and modeling novices (students) in order to observe possible differences in collaboration behaviors and draw conclusions regarding our approach. While students received an artificial problem and modeling task description, practitioners were asked to tackle a problem they were currently working on in their company.

An aspect not covered by these research questions is the quality of the resulting metamodels. Our goal was to make the transformation of sketches into models possible, and not to measure metamodel quality. Investigating and measuring the quality of the metamodels produced by our approach is out of scope for this thesis, but could be future work. In particular, an interesting question would be whether the generated metamodels are useful in subsequent project phases. It would be necessary to define quality criteria – such as completeness, complexity, level of abstraction in relation to the metamodel's purpose [Jea13], its ease of use, and its suitability for re-use – and to evaluate the metamodels according to these criteria in longitudinal case studies.

1.4 Overview of Our Flexible Modeling Approach

Our flexible modeling approach combines the flexibility of unconstrained sketching with the power of semi-formal modeling, and provides requirements engineers with flexible tools for early requirements modeling. Its focus lies on simple diagrams consisting of nodes and edges. The approach enables drawing of free-form diagram sketches and models with various degrees of formality, and supports a step-wise transformation of sketches into models. The idea is that the free-form sketching part mimics physical media (such as whiteboards and paper). A tool implementing our approach should have an unobtrusive interface and should not impose any specific process or workflow, such that users do not get distracted and can concentrate on their creative requirements modeling task.

We have embodied our approach in the FlexiSketch tools. They consist of FlexiSketch for Android devices, and FlexiSketch Desktop for computers and electronic whiteboards. Figure 1.4 shows screenshots of the tools. As shown in Figure 1.5, FlexiSketch focuses on three activities:

1. Sketching/Modeling. Users can perform free-form sketching. Their strokes get converted into distinct symbols and links, which makes it possible to create diagrams consisting of nodes and edges. Technically, nodes and edges are handled as different elements.

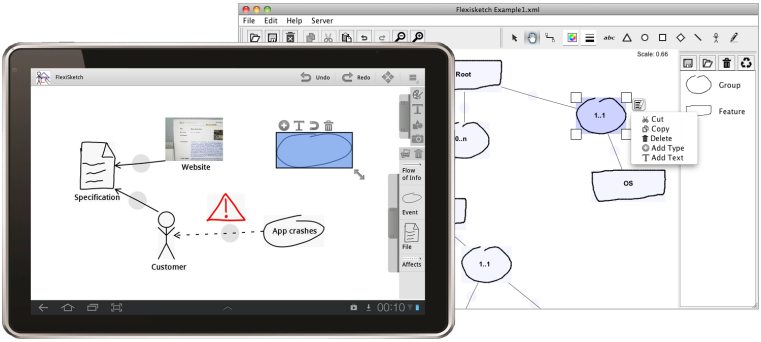


Figure 1.4: The FlexiSketch tools (with the Android version on the left, the desktop version on the right).

2. Metamodeling. Our approach contains a lightweight end-user metamodeling method. This method allows users to define their sketched constructs by assigning meaning to them, and to define cardinality rules for links. Where user-defined cardinality rules are missing, our approach automatically infers cardinality rules by using the *closed world assumption*. In contrast to related work that focuses on (full-fledged) metamodel creation, we follow an approach of "just enough metamodeling": users of our approach should have to perform as few metamodeling actions as possible, while still being able to export their sketches as models. We incorporate the lightweight metamodeling mechanisms in such a way that they are as invisible as possible to the user. However, our approach is also useful for engineers whose goal it is to define preliminary, simple modeling languages. The outcome of this activity is typically not a high quality modeling language,

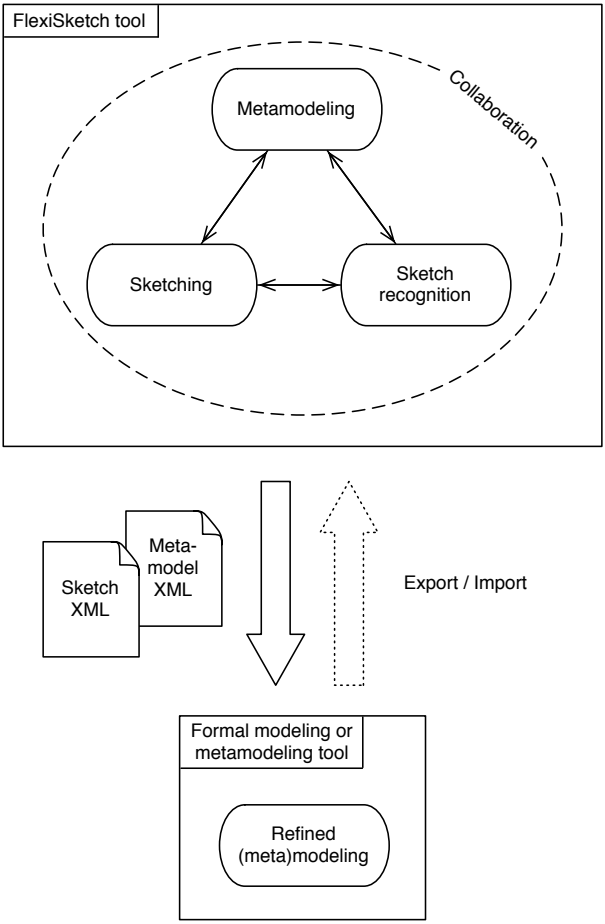


Figure 1.5: High-level scheme of our approach.

but a simple metamodel construct that can later be refined further by a language designer if needed.

3. Sketch recognition. Our approach makes use of existing sketch recognition technology to identify similarities between already defined constructs and newly drawn elements. This relieves the users from the effort of defining every instance of the same modeling construct individually. Our approach uses on-line sketch recognition, meaning that it recognizes elements while the user is drawing. It cannot perform off-line recognition, which means to recognize individual elements in an already finished model sketch.

Our tools also support a collaborative mode: multiple users can connect their individual mobile devices to the desktop version (which acts as a server), and simultaneously work in the same workspace. They can sketch together and perform collaborative, lightweight metamodeling. More details about the tools are provided in the individual chapters (e.g., see Chapter 2 for the mobile version and Chapter 3 for the desktop version).

Our approach combines metamodeling and sketching/modeling in a single tool. It can bridge the gap between sketches and (semi-) formal models to varying degrees, depending on how much metamodeling is performed by the users. The amount of metamodeling that is needed in turn depends on how users would like to re-use and export the created artifacts. Our approach supports the creation of user-defined lightweight modeling languages. At the same time, the metamodeling part is completely optional; users can

also decide to use our approach for pure sketching. Regarding the export, we can distinguish between four levels of formality, whereby the first two levels do not need any metamodeling:

- Sketch image file. This is the least formal export option, storing the created artifacts as an image file.
- Node-and-edge model. The created artifacts are exported as an XML file. The format distinguishes between generic symbols and generic links, and lists them as individual elements. Users can choose this option if they want to re-use the created artifacts in other, classic modeling tools that support generic diagrams (such as MS Visio[3], Draw.io[4], or yEd[5]). This option only needs a parser to parse the XML file into a format that is understood by the respective modeling tool.
- Specific model type. Before exporting the created artifacts, users assign specific types to all elements that should be exported and make sure that the artifacts adhere to a particular, existing modeling language (e.g., UML use case diagram). This option is useful when users start to sketch their ideas without adhering to any particular modeling language, and decide later on to convert their sketch into a more formal model (e.g., users could first draw some generic entities and relationships between them, and decide later on that the sketch should be enhanced and turned into a UML class diagram). Our approach creates two XML files, one for the

[3]https://products.office.com/en-us/visio/ [last checked: 12/04/15]
[4]https://www.draw.io/ [last checked: 12/04/15]
[5]https://www.yworks.com/products/yed [last checked: 12/04/15]

model sketch, and one for the metamodel. Alternatively, if users already know from the beginning what modeling language they want to use, and if the respective metamodel already exists, they could load the existing metamodel right away and use the given language constructs for creating their sketch. Then, a parser can be used to parse these files into a format that is compatible with the target tool (e.g., ArgoUML[6], Enterprise Architect[7], or the Eclipse Modeling Framework[8]). Assuming that the target tool contains a formal description of the used modeling language, the models can be imported and are amenable to further processing such as model simulation, model checking, and automated source code generation (the features depend on the target tool).

- Model with custom metamodel. This way of exporting artifacts is basically the same as the previous one, but in this case the chosen modeling language does not exist. Thus, in order to import the artifacts in another tool, that tool must be able to handle the import of new metamodels. The purpose of such tools is usually the creation of metamodels (e.g., Eclipse Ecore[9], Adoxx[10], or MetaEdit+ [KLR96]), and they are called metamodeling tools. To make the output of our approach compatible with a metamodeling tool, a parser must be written that translates the format of our metamodel according to the format given by the target tool's meta-metamodel (or in other words, a transformation be-

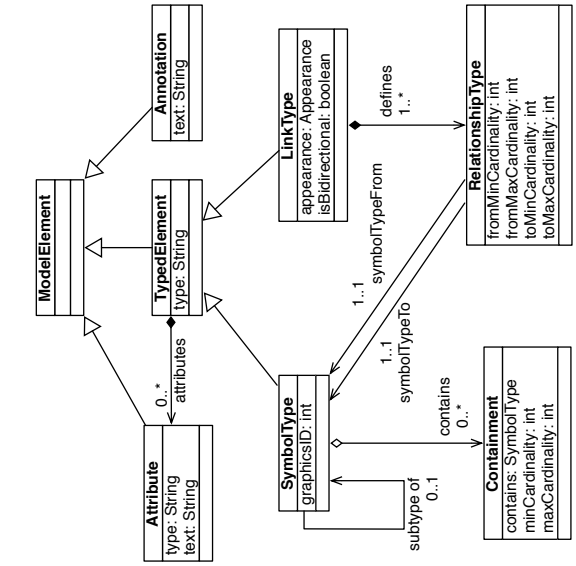
[6] http://argouml.tigris.org/ [last checked: 01/12/16]
[7] http://www.sparxsystems.com.au/ [last checked: 12/05/15]
[8] https://eclipse.org/modeling/emf/ [last checked: 12/05/15]
[9] https://eclipse.org/modeling/emf/ [last checked: 12/04/15]
[10] https://www.adoxx.org/live/home [last checked: 12/04/15]

tween our meta-metamodel and the meta-metamodel of the target tool must be specificed). This must be done only once for each target metamodeling tool, afterwards it is possible to import all metamodels created with our approach.

Since our approach allows for flexible sketching where users do not need to adhere to predefined modeling languages, exporting custom metamodels together with models is the most interesting export option for us. As a proof of concept, we have developed a parser to export our artifacts to MetaEdit+, a commercially available metamodeling tool [KLR96]. Figure 1.6 describes our meta-metamodel. More details about the meta-metamodel can be found in Chapter 4.

For the export, we translate and store the user-created metamodel according to the GOPPRR format [KHK11], which is used by MetaEdit+. Once the metamodel and the model are imported in MetaEdit+, the user can start to refine them. MetaEdit+ accepts changes to the model if they comply with the metamodel, i.e., it checks the cardinality constraints and does not accept new connections that were not defined in FlexiSketch, unless the user adds them to the metamodel. Figure 1.7 shows a model using a custom modeling language created in FlexiSketch, and its exported version in MetaEdit+.

As shown in Figures 1.5 and 1.8, our approach also includes the scenario of importing refined models back to FlexiSketch, which is theoretically possible as long as no new and incompatible metamodel elements are added. However, our approach is meant to




```
Metamodel = {ModelElement};
ModelElement = TypedElement | Annotation | Attribute;
TypedElement = SymbolType | Relationship;
SymbolType = Shape | Picture;
```

```
SymbolType
    type: String
    graphicalRepresentation: int (id to shape or picture)
    attributes: List<Attribute>
    isSubtypeOf: SymbolType
    containments: List<Containment>
        Containment
            contains: SymbolType
            minCardinality: int
            maxCardinality: int
LinkType
    type: String
    appearance: Appearance
    isBidirectional: boolean
    attributes: List<Attribute>
    relationships: List<RelationshipType>
        RelationshipType
            symbolTypeFrom: SymbolType
            symbolTypeTo: SymbolType
            fromMinCardinality: int
            fromMaxCardinality: int
            toMinCardinality: int
            toMaxCardinality: int
Attribute
    key: String
    value: String
```

Figure 1.6: Overview of our meta-metamodel in textual form and as UML class diagram.

support early design and idea generation activities. Once models sketched with FlexiSketch are exported and refined, there might be other modeling tools that are more suited to continue the further development of the models. Therefore, we consider the scenario of importing artifacts back to FlexiSketch to be less relevant compared to the scenario of exporting artifacts, and thus it has not yet been implemented in our tools.

The overall process of working with model sketches when using FlexiSketch is depicted in Figure 1.8.

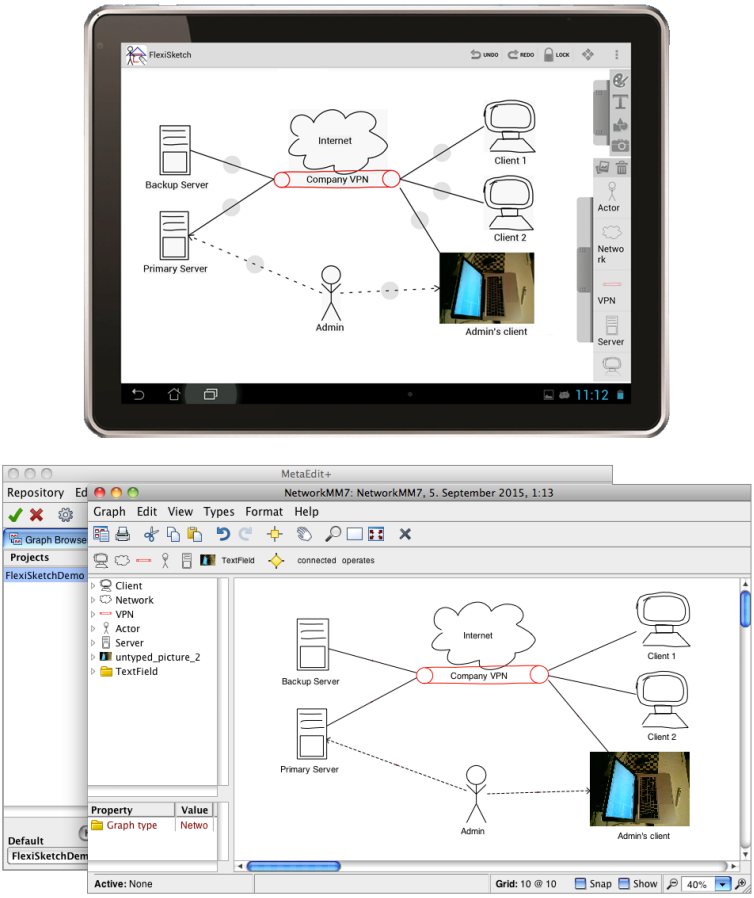


Figure 1.7: A model and its underlying custom modeling language were exported from FlexiSketch to MetaEdit+.

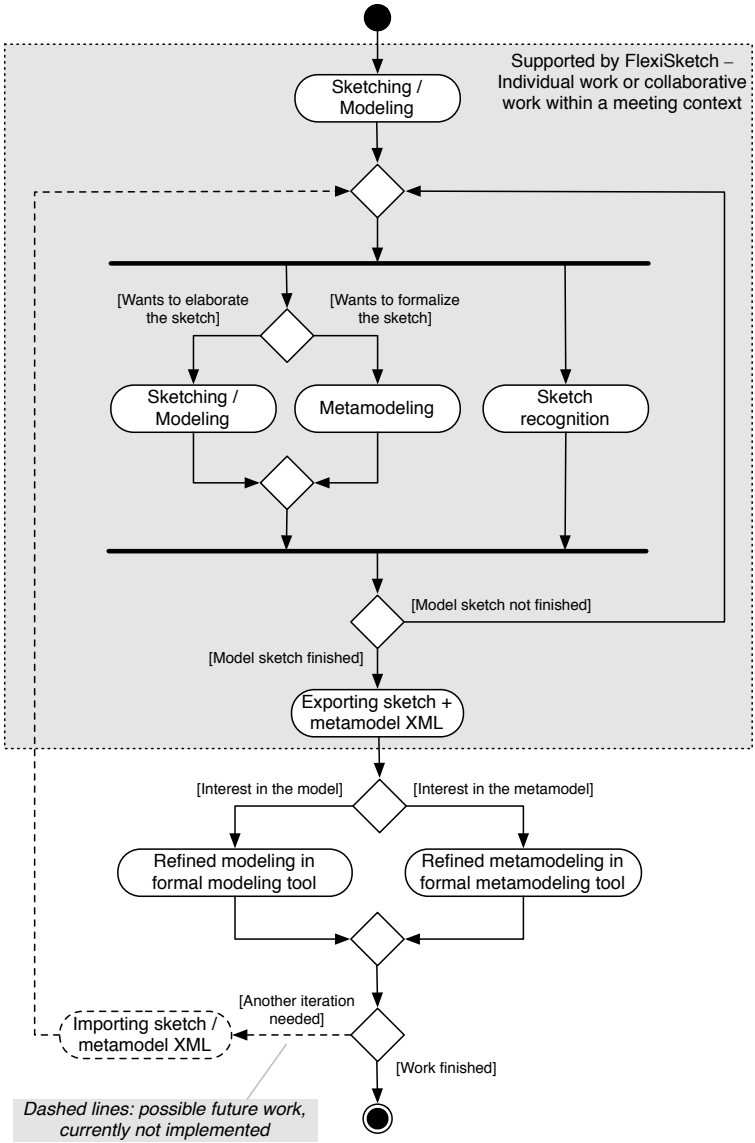


Figure 1.8: A UML activity diagram shows how our approach can be used.

1.5 Contributions

The main contribution of this thesis is a flexible modeling approach that combines the flexibility of free-form sketching with the power of semi-formal modeling. The approach makes it possible to start with completely informal drawings and to transform them into semi-formal models step-by-step. By doing this, the approach overcomes the media disruption between sketches and graphical models. The thesis makes the following contributions:

- The conceptual solution of our approach that shows how free-form sketching and structured modeling can be combined in a single environment;
- A method for lightweight end-user metamodeling in a sketching environment;
- The FlexiSketch Android and desktop tools that implement our approach. They fill a gap in the current modeling tool landscape that exists between semi-formal modeling tools and completely free-form sketching tools; and
- An evaluation of our approach consisting of multiple experiments and studies that assess the utility and usability of our approach.

1.6 Research Methodology

Our research methodology is based on methods published by Wieringa and Heerkens [WH06], and Easterbrook et al. [ESSD08].

The research questions implement the six steps of the *engineering cycle* [WH06]: *problem investigation, solution design, design validation, choose a solution, implement the chosen solution,* and *implementation evaluation.* They represent an *exploratory question* (RQ 1), a *design question* (RQ 2), and two *evaluation questions* (RQ 3 and RQ 4) [ESSD08]. RQ 1 addresses a *knowledge problem* [WH06], since we want to know whether the gap between sketches and models does indeed affect engineers in their daily work, and whether there is room for improvement. RQ 2 then addresses a *world problem* [WH06]: we want to design a solution that improves the state of the world. RQ 3 and RQ 4 address knowledge problems – we want to find out how successful our designed solution is.

We did not perform the steps of the engineering cycle in a sequential manner, but rather used an iterative process. We iteratively evaluated and evolved our approach, similar to how a software system grows in an evolutionary software process model. This enabled us to receive early feedback on our ideas and tool prototype, and to enhance our approach accordingly.

In particular, we created a first prototype of our idea, and in parallel set up the first study to find out why engineers use paper and whiteboards, and what needs they have. The study contained semi-structured interviews with practitioners. At the end of each session, we showed them our initial tool prototype to get first feedback and to discuss practitioners' needs in more detail. The data from this study was used to refine our approach and to develop new prototypes, e.g., the desktop version and the collaborative version of FlexiSketch. These tool versions were then used to evaluate our approach in more detail.

1.7 Chapter Summary

The remainder of this thesis consists of a collection of scientific publications. Chapters 2 to 6 each are a paper and contribute to the thesis goal, but are also contributions on their own, while Chapter 7 summarizes and concludes the thesis. The papers in Chapters 2 to 5 are peer-reviewed and published. Chapter 6 contains a working paper submitted for a journal publication. This section presents the thesis roadmap and summarizes the chapter contents.

The roadmap is shown in Figure 1.9. The figure provides an overview of the relations between the chapters and the research questions from Section 1.3. Each color represents one thesis chapter. Chapter 2 presents a holistic overview of our approach by explaining its three parts (*modeling*, *metamodeling*, and *sketch recognition*), the first tool prototype, and reports on initial evaluations regarding the feasibility and utility of our approach. Then, Chapter 3 presents the technical aspects of our tool in more detail, including tool extensions for collaborative work. Chapter 4 enhances our metamodeling concept. Finally, Chapters 5 and 6 evaluate our approach. Chapter 5 focuses on collaborative sketching and language definition. Chapter 6 extends Chapter 5 and reports the results of a study about end-user metamodeling.

Chapter 2 presents our initial conceptual and technical solution for FlexiSketch, addressing RQ 2 in a preliminary way, and reports on two experiments for evaluating the feasibility and utility of

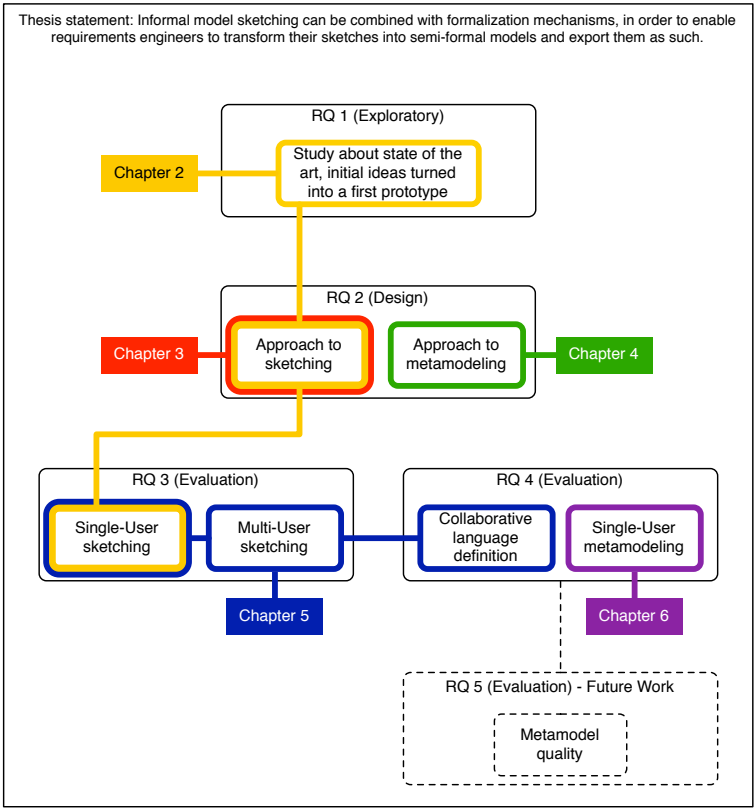


Figure 1.9: Thesis roadmap: the mapping between papers and research questions.

our approach (RQ 3). Our approach proposes a method where a software tool mimics an empty sheet of paper, and users can switch between free-form sketching activities and metamodeling activities at any time. We performed the experiments with students and practitioners. The experiment sessions were preceded by semi-structured interviews to obtain an answer to RQ 1: we asked the participants how they sketch models in their daily work or studies, how often they do this, and how they re-use the sketched information. Answers from participants show that they indeed use paper and whiteboards, and that the modeling notations they use resemble standard notations, but deviate from them. Although sketches are frequently discarded and not re-used, almost all participants reported that they experience situations where they go through different amounts of effort to re-create sketches as models or to copy information contained in sketches. After we presented our prototype and performed the experiment, they agreed that our approach could be useful in these situations. While the experiments confirmed the feasibility, we gathered a lot of usability-related feedback that we were able to incorporate in later tool prototypes.

Chapter 3 extends our approach with a collaboration concept. It focuses on the technical enhancements and presents our latest tool prototype, FlexiSketch Team, addressing RQ 2. This solution consists of a desktop application (FlexiSketch Desktop) and an enhanced version for Android devices. Our concept supports collaborative work with a multi-screen setup: multiple tablets can be connected to the desktop application which acts as a server. Users have simultaneous access to the same workspace

and can sketch and define modeling languages collaboratively. FlexiSketch Team synchronizes user actions and provides some basic collaboration features such as a locking mechanism that prevents conflicting inputs. Apart from the collaboration support, FlexiSketch Desktop can also be used as a standalone version on PCs and electronic whiteboards.

Chapter 4 proposes a lightweight metamodeling method for end-users (i.e., requirements engineers without advanced metamodeling knowledge). It describes how users can define their own modeling languages by performing metamodeling activities at any time during model sketching. In the background, FlexiSketch builds a simple metamodel while the users are sketching. The method combines metamodel information that is manually entered with automatically inferred data: some information comes from the users, e.g., when they define symbols or cardinality rules. Other parts of the metamodel are automatically inferred by the tool, e.g., cardinality rules are computed by analyzing the model sketch as long as the user does not enter cardinality rules manually. To ensure metamodel completeness, the user can consult a wizard that goes through undefined model constructs and asks the user to enter missing metamodel information. The sketches and the corresponding metamodels can be exported as xml files at any time.

Chapter 5 reports on a qualitative study involving groups of students (modeling novices) and practitioners (experienced requirements engineers). By investigating how three groups of novices and three groups of experienced modelers define modeling languages

collaboratively when using our tool, its focus lies on answering RQ 4.2. However, since metamodeling is closely tied to sketching in our approach, the study also provides answers to RQ 3. The main outcomes of the study are: i) simultaneous sketching happened in all groups and alternated with discussion phases, ii) all participants took an active part in the metamodeling activities, and iii) while practitioners communicated well with each other, students had some coordination problems because their attention was drawn to the tool (i.e., their individual tablet screens). Results suggest that it is preferable to have one single, big screen for all collaborators (i.e., using FlexiSketch Desktop on an electronic whiteboard) in situations where this is feasible. Working with multiple screens is an alternative if no such big screen is available, or if the collaborators are geographically distributed. For the latter case, our tool should be extended with user awareness features.

Chapter 6 extends Chapter 5 with a quantitative experiment, and addresses RQ 4.1. We investigated how well individual novice and expert modelers are able to enter correct metamodel information with the mechanisms provided by our approach. This is important to know because our approach partly relies on this information for building the metamodel. The results of this experiment show that, on the one hand, experienced modelers have no problem using our lightweight metamodeling approach, and some of them would even like to see more metamodeling options combined with the sketching environment. On the other hand, novice modelers sometimes have difficulties to distinguish between the modeling and metamodeling level, especially when they are supposed to define cardinality rules for links. To better support novice modelers, it

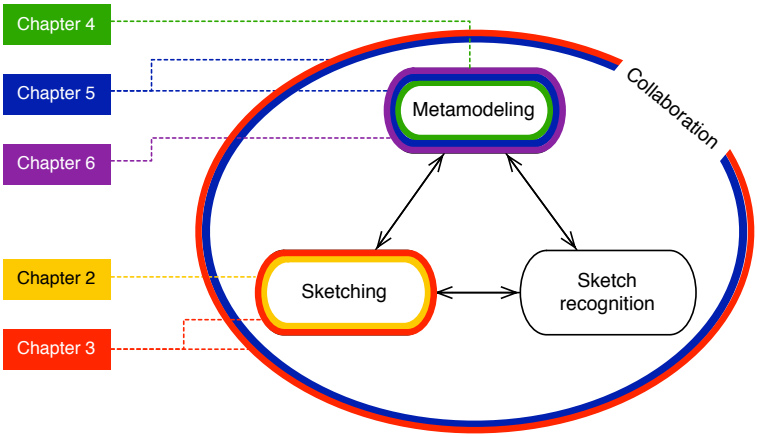


Figure 1.10: The main focus of each chapter.

would be worthwhile to think about how additional metamodeling guidance methods can be incorporated into our approach.

Chapter 7 concludes the thesis by summarizing the contributions and possibilities for future work.

Figure 1.10 shows the main focus of each chapter from 2 to 6. While Chapters 2 and 3 focus on individual and collaborative sketching, Chapters 4 to 6 focus on individual and collaborative metamodeling. This distinction is made to provide an alternative overview of the thesis chapters, it does neither imply that Chapters 2 and 3 are solely about sketching, nor does it imply that Chapters 4 to 6 are solely about metamodeling. While the foci of the chapters differ, they also discuss the other parts of the approach.

Chapter 2

FlexiSketch: A Mobile Sketching Tool for Software Modeling

Original publication:
FlexiSketch: A Mobile Sketching Tool for Software Modeling
D. Wüest, N. Seyff, and M. Glinz
International Conference on Mobile Computing, Applications and Services 2012

Abstract

Although most software engineers have access to various modeling tools, they often use paper and pencil to sketch ideas and to support modeling activities. This is particularly true when they are working in the field, for example gathering requirements from stakeholders. Sketches documented on paper very often need to be re-modeled

in order to allow further processing – an error-prone and time-consuming task. The aim of our work is to better integrate these early sketching and modeling activities into the overall software engineering process. We have prototyped FlexiSketch, a mobile application that supports free-form, flexible, in-situ modeling and allows software engineers to annotate their models. Apart from the application and the underlying conceptual solution we also present the results of initial experiments. Those suggest that the tool supports free-form sketching similar to paper and pencil, and that practitioners would be willing to use a tool like FlexiSketch in their daily work.

2.1 Introduction

Early software engineering (SE) activities include gathering and documenting needs that stakeholders of a future software product have. Also, first design sketches of the future software system are created. These tasks often ask for creativity. Diagram-like sketches (consisting of nodes and edges) can help to depict current systems and communicate ideas in a simplified way.

Engineers and stakeholders mostly prefer to use paper and pencil, whiteboards, and flip charts for sketching in early SE phases [GD96, BGCB10]. This is mainly due to two reasons: First, engineers and stakeholders meet in various places. Paper and pencil, or flip charts, are available anywhere and are instantly ready for use. Second, they are easy to use. They allow for informal, unrestricted sketches. In contrast, most SE modeling tools follow a more formal modeling approach [OBS+10] and require an engineer to use a specific modeling language and syntax.

Once relevant ideas are documented on paper, the question is how to store, distribute, and make the information amenable for further processing. One option is to take photographs of sketches and later, back in the office, re-model the information manually with the help of a software modeling tool. This media break is error-prone and the re-modeling task time-consuming. Information may get lost or be interpreted in a wrong way. This is particularly a risk when another person is responsible for the re-modeling task or information is transcribed after some time has passed by.

To avoid the need for re-modeling information manually, we propose that software engineers use lightweight software tools right from the beginning of a project. Such tools must have a similar availability to that of paper and pencil-based approaches and allow for free-form sketching. In our current research, we focus on multi-touch mobile devices such as tablet computers to support these sketching activities. Today, such devices are widespread, ad-hoc available, and have sufficient computing power. Therefore, we consider such devices to be an ideal platform for supporting engineers with informal sketching and modeling tools which also allow them to stepwise refine and formalize their sketches.

In this paper we present a tool-supported approach that provides users with free-form sketching capabilities and allows them to annotate their sketches, thereby defining their notations and enabling a semi-automatic formalization of the sketches. Furthermore, we elaborate on the results of two experiments that focused on qualitative feedback regarding the usability and utility of our approach.

The remainder of this paper is structured as follows: We present our research goal and approach in Section 2.2. The tool prototype is described in Section 2.3. The two experiments and results are discussed in Sections 2.4 - 2.7. Section 2.8 presents related work, and Section 2.9 presents conclusions and future work.

2.2 Flexible Sketch-Based Modeling

2.2.1 Main Goal

The overall goal of our work is to unite the flexibility of unconstrained sketching with the power of formal modeling. In this context, we want to provide a tool-supported approach that (i) allows users to sketch any informal models, (ii) provides means for assigning syntax and semantics to sketched elements on the fly, and (iii) supports the semi-automated transformation of sketches into classic semi-formal models (e.g. a class diagram or a statechart) [WG11]. We envision our approach to allow for free, flexible sketching (as pen and paper does), and at the same time support a step-wise beautification and formalization of the sketches, thus avoiding the media break between early software engineering sketches and semi-formal models [Wüe11].

2.2.2 Key Requirements

Discussions with experts and the study of related work led to the following high-level requirements that we consider to be relevant for building a solution that fulfills end-user needs:

High flexibility: Users shall be able to sketch any kinds of diagrams. There should be no restrictions limiting users' expressiveness.

Natural sketching: The tool shall allow the use of input devices that give users a natural feeling for sketching, for example, a tablet PC with a pen or an electronic whiteboard.

Formalization capabilities: The tool shall support the transformation of diagram sketches into classic semi-formal models by, e.g. a semi-automated method, thus avoiding media breaks.

Speed: The tool shall allow fast creation and annotation of sketches. If the process is not faster than traditional sketching followed by model re-creation in a modeling tool, nobody would be motivated to use it.

The biggest challenge regarding our work is the aim to give users maximum flexibility in what they are sketching, not restricting them to any specific language or notation. Yet, a tool needs to "understand" what a user is sketching in order to perform any transformation of sketches into semi-formal models. We address this challenge by giving users the freedom to draw anything they want, and request that users assign meaning to sketches in due course with the help of annotations. These annotations will then be turned into corresponding metamodel elements.

2.2.3 Our Approach

We aim at a process that lets users switch arbitrarily between three work modes (Fig. 2.1): in the *modeling* mode, the user creates, augments or modifies sketches. In the *metamodeling* mode, the

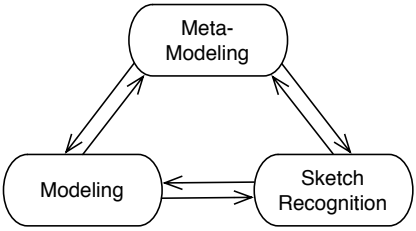


Figure 2.1: The three activities leading to a semi-formal model in the end.

user annotates sketches and the tool creates metamodel elements based on these annotations. In the *sketch recognition* mode, the tool transforms sketched elements into semi-formal model elements by recognizing and interpreting sketches based on the information currently available in the metamodel. To make this process work, annotating must be easy and straightforward. No programming, scripting, or metamodeling skills should be required for this task. This free interleaving of modeling, metamodeling and recognition activities is the key difference between the presented approach and related work, where any form of metamodeling has to be done first. The following paragraphs depict our vision in more detail.

Modeling is facilitated by two drawing modes. One mode mimics a whiteboard and allows for free sketching, while the other mode enables users to modify individual symbols (e.g. scale, move, copy). As opposed to other work, we do not define the modes as exclusive modes, i.e. the modes are not switched by pressing a dedicated button. The modes are switched implicitly. We believe that a single mode stays closer to the pen and paper metaphor. As

soon as a user selects an already sketched object, she can make modifications.

Metamodeling starts when users assign a type to a previously sketched symbol. The symbol then gets added automatically to a dynamic symbol library. A symbol library consists of all defined symbols including their types, and allows to decouple types and their graphical representations. Symbol libraries are the first part of a lightweight, end-user metamodeling approach that supports users in defining a modeling language. Each symbol library contains a set of symbols belonging to a specific context, i.e. a specific diagram type. Users can define multiple contexts where the same symbol has different meanings. Because people like to mix different visual conventions during creative tasks [CVDK07], multiple symbol libraries can be active at the same time. A big challenge is to invent an easy-to-use interface for more complex metalanguage definitions, such as associations and rules for associations.

Sketch recognition happens in an interactive manner. Users are encouraged to take part in the recognition process to train the recognizer. They are also able to decide when and if recognition feedback should be given or not. These options assure that users do not get distracted from their sketching.

Our flexible process produces two kind of re-usable artifacts: the models drawn by the user, and a partial metamodel that is contained in the created (or re-used) symbol libraries. We envision that these two artifacts will make it easy to share models between different persons. Moreover, exporting the sketched models and

further processing them in more formal SE tools comes within reach.

2.3 The FlexiSketch Prototype

We developed the FlexiSketch prototype, a mobile tool for free-form sketching with the focus on diagram sketching for SE. The prototype serves as proof of concept that the idea of letting the user switch flexible between all three activities works. In this section we present our prototype in detail, and we show how it realizes each of the three activities depicted in Figure 2.1.

Our tool is written in Java using the Android SDK for the Android OS, supporting version 3.0 and above[1]. We especially focused on tablet computers (with screen sizes around 10 inches) since we consider smart phone screens to be too small for effective sketching. Figure 2.2 shows a screenshot of our prototype.

2.3.1 Modeling

In order to emphasize the possibility of free-form sketching, the tool shows a white drawing canvas (1) when it is started. Additional functionality is hidden in the action bar at the top (2) and pull-out

[1]FlexiSketch is available on Google Play (formerly known as Android Market). A video demonstration can also be found on Google Play or at http://www.youtube.com/watch?v=D06t0K5Otzw

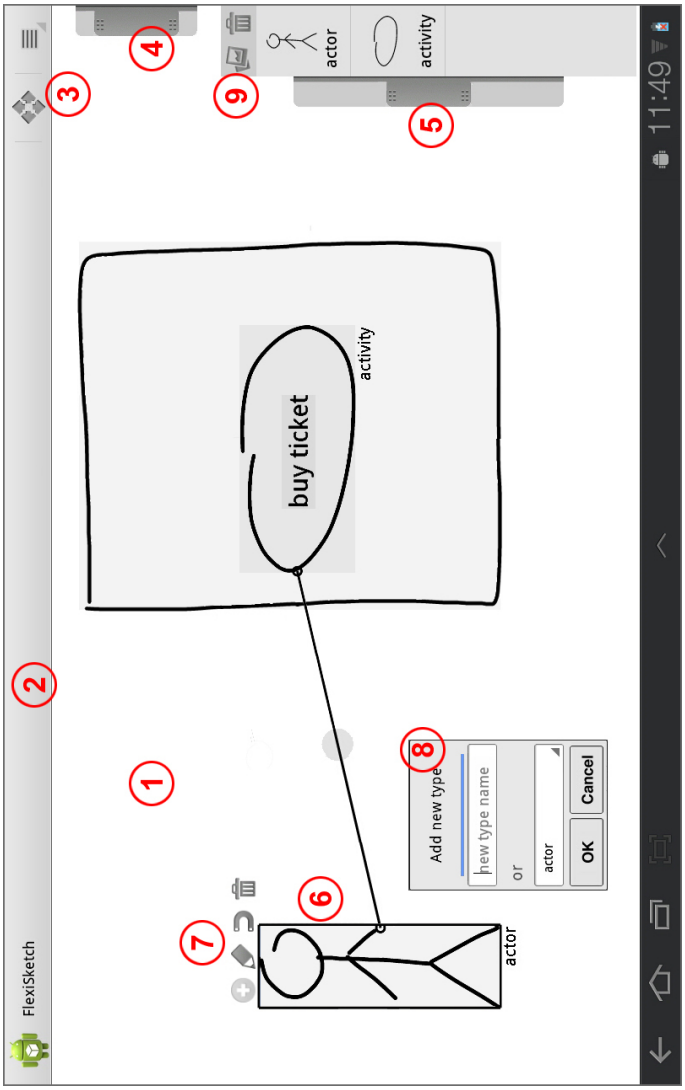


Figure 2.2: Screenshot of the FlexiSketch prototype.

containers (4,5) on the right edge of the screen. To not distract the user from free-form sketching, we wanted to hide as much of the GUI as possible. However, we included a permanent action menu bar at the top of the screen to make some functions like scrolling quickly accessible. Scrolling of the drawing canvas is activated at the push of a button (3), and allows the user to draw sketches in the size of an A4 paper, independent of the actual screen size of the mobile device.

The user can start to draw like she would in any other paint application, either with her fingers or with a stylus. When the user stops to draw a particular item and removes the fingers from the screen for a predefined amount of time, the drawing is converted into a distinct symbol. The conversion of drawings into a symbol is not visible for the user per se, but the tool colors the background of a symbol with a light gray to give feedback that the conversion took place.

Symbols can be selected by tapping on them (touching the screen and lifting the finger without moving it). A selected symbol (6) is highlighted with a rectangular border around it and some context menu icons on top of it (7). The symbol can then be manipulated, dragged around, or deleted. The tool does not provide two distinct modes for drawing and symbol manipulation in order to allow for free-form drawing at any time. If the user touches the screen outside of the selected symbol and moves the finger, i.e. on white space or another symbol, she can just continue to draw naturally.

When the user starts sketching on top of one symbol and ends the stroke on top of another symbol, the tool recognizes the drawing as an association between the two symbols. It replaces the drawing by a straight line. In the middle of the line there is an anchor point that allows to select the line (otherwise the line would be hard to grab) and show its context menu. The line replacement is optional and can be switched off in the options menu.

An arbitrary amount of text boxes can be attached to symbols via the context menu (7). The text boxes are tied to the symbols, but can also be moved individually.

2.3.2 Metamodeling

The prototype provides means for defining symbols by assigning types to them. This is the basic functionality needed for the symbol library to work. After sketching a symbol, the user can choose to define its type via the context menu (7). A small popup menu provides the user with a list of all types that she already created, and an option to add a new type (8).

When the user decides to define a new type, she enters a name for it with the standard virtual keyboard. The tool creates a new entry in the symbol library with the given name, and stores the selected symbol in it. The symbol library contains entries for all defined types, and each type is graphically represented by at least one symbol (the symbol that was selected when the user defined the type). A list consisting of these graphical representations is

shown in the pull-out container at the right edge of the screen (5). From there, defined symbols can be re-used with a drag-and-drop gesture.

The symbol library allows to add multiple symbols to the same type. This flexibility is needed because different users might draw different symbols to depict the same meaning. The opposite can also happen: the same symbol might have different meanings in different contexts, i.e. in different diagram types. Therefore, a symbol library is meant to be a collection of symbols belonging to a single diagram type. Symbol libraries can be stored and loaded (9) independently from the user sketches.

Theoretically, associations can have different types (similar to symbols), but in the current prototype, the number of association types is limited and bound to predefined line styles. The look can be quickly changed via a drop-down menu.

To help the user to not forget defining the symbols, the main menu (2) includes an option to visually highlight all symbols on the screen that do not yet have a type assigned.

Types assigned to symbols can be shown or hidden through the options menu. This allows the user to show types while she is taking care of type definitions. She can hide the types when she is working with text boxes, which do not belong to type definitions, but to particular instances of types. For example, in a class diagram most symbols are of the type class, but each symbol on the screen has a different class name written in a text box. To

reflect the distinction between symbol types and text content of symbols, the functionality is strictly separated, accessible through two different context menu icons.

2.3.3 Sketch Recognition

We distinguish between *object recognition* and *sketch recognition*. We define object recognition as the automatic process of dividing user drawings into distinct symbols (and associations). The prototype does this while the user draws by using a simple timeout mechanism.

We define sketch recognition as the automatic process of comparing symbols to each other in order to identify similar symbols. For example, the user draws a stickman, and assigns the type *actor*. The sketch recognizer detects when the user draws a second stickman, so that the tool can automatically assign the type *actor* to it.

The prototype performs sketch recognition on drawn symbols. After the user draws a symbol, the sketch recognition algorithm searches the symbol library whether there are similar, already defined symbols. If the algorithm does not find any similar symbol, nothing happens. If a similar symbol is found, a pop-up on the bottom of the screen presents the symbol type for several seconds. If multiple similar symbols are found, the three closest matches are shown. The user can tap on one of the proposed types to assign this type to the newly drawn symbol. Choosing one of the

proposed types is optional. The user can also decide to ignore the proposals and continue sketching. In this case the proposals will fade out again. This mechanic allows to give non-disruptive sketch recognition feedback to the user. If the user chooses one of the proposals, the new symbol is added to the respective type entry in the symbol library. In this way we realize a trainable sketch recognizer that learns from user inputs. The training is transparent to the user in order not to distract her from her actual task.

When the user draws a symbol that looks similar to an already defined symbol, an optional feature allows to replace the newly drawn symbol by the defined one. This can help to get a more uniformly looking sketch at the end. The feature can also be used to beautify symbols: A user can place one of the provided standard geometric shapes in the drawing canvas and assign a type to it. When she then draws a similar symbol by hand, the sketch recognition mechanism proposes to replace it with the geometric shape. Figure 2.3 shows a use case diagram where sketch recognition was used to beautify the hand drawn symbols.

For the implementation of the sketch recognition algorithm, we adapted an algorithm that is based on the Levenshtein string distance [CSVV07], which calculates the distance between two strings.

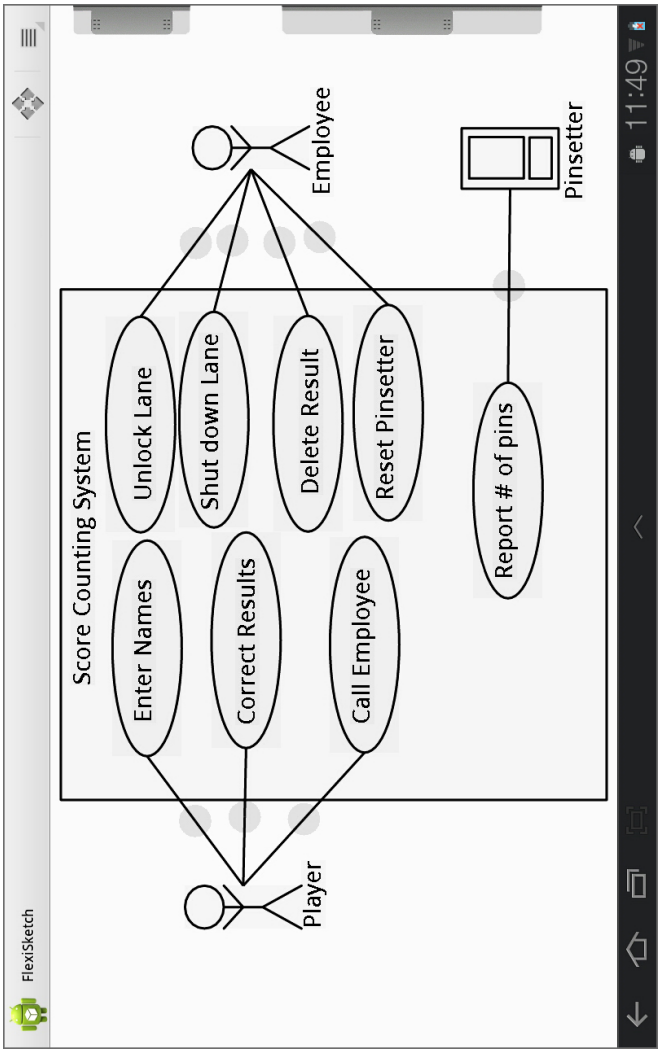


Figure 2.3: Sketch recognition beautified a hand drawn use case diagram.

2.4 Evaluation

We used the developed prototype to evaluate our approach in early 2012. We performed two experiments to assess the usability and utility of our tool-supported approach. The perceived utility, or usefulness [Moo03], affects the intention to use the approach, and therefore influences how much the approach will be adopted in practice [MSBS03].

We especially wanted to know whether our tool provides the same kind of flexible, ad-hoc sketching support that is also provided by paper and pencil-based approaches. Therefore we asked: What are the differences between FlexiSketch and paper and pencil-based approaches (*RQ 1*)? We also wanted to know whether users can in general classify the elements they draw with our tool into types (*RQ 2*). For adoption in practice, a tool has to be tailored to the needs of its users. Therefore we asked: Can they sketch diagrams with our prototype well enough such that they would consider adopting our tool in practice (*RQ 3*)? This also included to investigate what kind of diagrams software engineers sketch in practice.

The goal of the first experiment was to identify usability issues and assess whether our tool and approach allow users to sketch diagrams. In this experiment, we focused on answering RQ 1 and RQ 2 for one particular diagram type: use case diagrams. We chose this diagram type because (i) use case diagrams are widely known (and thus can serve as a common basis for our evaluation),

(ii) they are typically used in early SE phases, and (iii) they are well-suited for transforming sketches into semi-formal models. This experiment did not allow to answer RQ 3.

The goal of the second experiment was to generalize from use case diagrams in order to answer RQ 1-3 for arbitrary diagrams. We wanted to investigate how the tool prototype can handle different diagram types (RQ 3). Another goal of the second experiment was to gain insights into how physical media for sketching are used in SE practice, and what kind of sketches practitioners draw.

In both experiments, participants used an Android tablet and a stylus to draw diagrams with our prototype and assign types to sketched symbols.

2.5 Experiment #1: Feasibility of our Approach and Tool Usability

This first experiment investigated the usability of our tool and the feasibility of our approach. The experiment was conducted in a controlled setting with 17 participants from research and industry: four undergraduate students and four PhD students in Computer Science, and nine software engineering practitioners, all having several years of experience (three persons counted as practitioners have returned from practice to academia). Practitioners were between 25 and 70 years old, and included experts in SE and HCI. 15 participants considered themselves to be novice users

regarding touch interfaces (using smart phones for standard tasks, having no or little experience with tablet devices). Two persons considered themselves to be expert users (having a more in-depth understanding of mobile devices and their features, which includes tablet devices).

2.5.1 Method

The evaluation was conducted with each participant separately. It included a short briefing, the actual evaluation where participants had to perform three tasks using our tool, and a debriefing. In the briefing, we first asked demographic questions about the participant's knowledge in SE, touch interfaces, and working experience. Then we presented our research and gave a short introduction to the tool (about three minutes). For the first task, participants were asked to sketch a use case diagram according to a given problem description (our sample solution depicted five use cases). As second task, participants had to create a type for at least one instance of every distinct symbol As a last task, participants had to sketch a second use case diagram from a problem description of similar size, but this time using their predefined symbols whenever possible. For the tasks, we asked participants to think aloud. Interaction between us and the participants was reduced to a minimum, but participants were allowed to ask questions. We recorded our observations. In the debriefing, we asked about the usability and utility of the tool. Questions were based on the IBM Computer Usability Satisfaction Questionnaire [Lew95].

The briefings lasted from five to ten minutes, working with the tool took between 20 and 30 minutes. Debriefings lasted from 30 minutes to one hour.

2.5.2 Results

Qualitative feedback from the participants and our observations revealed that they were able to draw use case diagrams with our tool. In debriefing meetings participants told us that they liked the informality provided by the tool. Furthermore, it took them a very short amount of time to learn how types are assigned to symbols and how the container with the symbol library is used (RQ 2).

When we asked about the symbol library that allows for the re-use of defined symbols, 14 out of 17 participants said that they like the feature and that they perceive it to be more or equally efficient than sketching by hand (RQ 1). Three participants argued that use case diagrams consist of simple symbols easy enough to sketch by hand each time. But most participants said that even if the re-use mechanism is slower, they prefer it over sketching for the reason that all instances of the same symbol look exactly identical.

Almost all participants stated that it is still a bit faster to sketch with pen and paper, but the additional functionality of the tool compensates for it (RQ 1). Most frequently mentioned advantages over paper and pencil were: the manipulation of symbols, the re-use

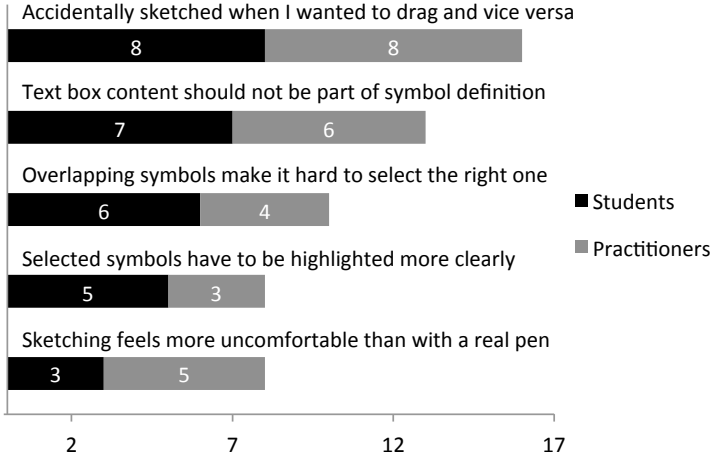


Figure 2.4: The 5 most frequently occurred usability problems.

of symbols, the save/load option, and the (not yet implemented) possibility to export the sketches to other programs.

The following features were also liked and mentioned frequently: (1) the possibility to draw anything, (2) sketched lines between symbols turning into straight connections, (3) the use of different colors to distinguish symbols, (4) the option to merge two drawn symbols into a single one, and (5) the text autocompletion feature provided by the OS.

In the following we discuss the most frequently mentioned usability issues (see also Fig. 2.4). Some of the issues are related to our design decisions and need re-thinking, while others are relatively easy to solve in future tool versions.

Sketching versus dragging: We do not provide two distinct modes for drawing and the manipulation of symbols. Participants often painted over existing symbols when they wanted to drag the symbols instead. Few said it actually makes more sense that the symbols have to be selected before they can be dragged around, but they continued to make the mistake.

Text boxes in symbol definitions: When users define symbols that contain text boxes, the text boxes - including the previously entered text - become part of the definition. This makes it possible to define, e.g. a class symbol with three text boxes. But participants said that it takes too much time to change the text of the text boxes: copies of defined symbols should come without, or with empty text boxes.

Overlapping symbols: Multiple taps on overlapping symbols successively select each symbol in turn. But participants told us that they did not figure this out and therefore had problems selecting symbols overlapped by other symbols.

Selection highlighting: Participants reported that the visual cue for highlighted symbols was not strong enough. As implication, they accidentally manipulated the wrong symbol, especially when symbols overlapped.

Sketching does not feel natural enough: Indeed, participants mentioned that sketching does not feel the same as with paper and pencil. The tip of the used stylus is wider than a normal pen, and the tablet has a tiny, but noticeable lag before it displays

Table 2.1: The 8 most frequently mentioned feature wishes.

	Students (8)	Prac. (9)	Total (17)
Larger screen (flip chart, whiteboard)	5	6	11
Symbol grouping for faster editing	5	5	10
Automatic beautification of symbols	5	5	10
Zoom / Scaling Function	3	7	10
Unlimited scrolling (larger canvas)	6	3	9
Resizing of drawn symbols	6	3	9
Landscape format	3	5	8
Ability to define types of associations	4	4	8

the drawn strokes. This is due to hardware characteristics of the tablet.

There are several feature wishes that participants mentioned a couple of times and therefore got a high ranking (see Table 2.1).

Larger screen: The wish for a larger screen (in the size of a flip chart or whiteboard) made it to the top. Related to this are the following wishes (see Table 2.1): a *zooming or scaling function*, *unlimited scrolling*, *resizing of drawn symbols*, and the ability to turn the tablet to use it in *landscape format*.

Symbol Grouping functionality: More than 50% of the participants would like to have a grouping function, so that they can manipulate a group of symbols at the same time (e.g. to move them around or to delete them).

Beautification: 10 out of 17 participants wish to have an advanced beautification function. In particular, they were suggesting to have

a feature that beautifies symbols based on their geometry. For example, a hand drawn rectangle should be displayed with straight lines, and a hand drawn oval should get smooth curves.

Different types of associations: Participants also stated the missing ability to define different types of lines, i.e. associations.

Further feature wishes include an undo feature, standalone text boxes and a distinct feature to add commentary text to the sketches.

2.5.3 Findings

With this experiment, we investigated whether the FlexiSketch tool can be used to draw use case diagrams, and whether it supports ad-hoc and flexible sketching. Furthermore, we investigated whether users are able to define the individual elements of a use case diagram by assigning types to symbols (RQ 2).

All participants confirmed that they could draw the diagrams corresponding to the given problem description. They liked how the tool allows for free-form sketching. While observing the participants during the modeling task, we noticed three problems that delayed the completion of the task. These issues were also mentioned by participants.

First, in several cases participants accidentally drew a stroke when they wanted to drag a symbol. We will have to investigate whether

this problem disappears once selected symbols get highlighted more clearly (which was also a mentioned critique). Therefore, we recently implemented a blue-colored background for selected symbols. We believe that users will associate the color with draggable symbols and will stop trying to drag symbols that are not colored blue. This is an assumption that yet needs to be evaluated. If the assumption does not hold, we have to rethink our design decision of not having two alternate modes for drawing and the manipulation of symbols.

Second, the screen size of the tablet computer proved to be limiting. Many participants lost time due to rearranging symbols, and they asked for a larger screen or an enhanced scrolling function. This seemed to be the biggest issue regarding adoption in practice.

Despite these issues, participants stated that drawing is easier and faster than with any other software modeling tool they know. Furthermore, we found the resulting diagrams to be well readable.

Participant feedback showed that they had no problem with defining the individual symbols (RQ 2). Some of them even started to define the symbols right away although we only asked them to draw a use case diagram as the first task. Around 50% of the participants, mainly students, asked how to make copies of symbols. Defining them is the only way in the current prototype. This might have been an additional motivation for the participants to do so.

Participants frequently mentioned the lack of certain tool features being a disadvantage, although physical media like paper and pencil do not have these features either (RQ 1). It seems that for paper and pencil, most people unconsciously accept the lack of symbol manipulation features. But they expect them to be present in a software tool. Participants stated that for them it felt a bit slower to draw with a software tool, thus a tool must compensate this by providing additional features. They mentioned that, once the formalization or export function is available, the whole process will be faster with our tool compared to paper and pencil.

2.6 Experiment #2: Utility of the Approach

This experiment gathered data about why and what kinds of diagrams practitioners sketch in their daily work, and assessed the utility of our approach. The experiment was conducted in a controlled setting with the nine practitioners from the first experiment. Thus, we refer to Section 2.5 for demographic information about the participants.

2.6.1 Method

The experiment was done with each participant separately, and included three parts: an interview, the evaluation and a debriefing.

We started with questions about (1) how participants use sketches in SE practice, (2) what kind of diagrams they sketch, and (3) how these sketches are re-used later on.

The participants were then asked to sketch an example diagram of a diagram type they use frequently in their daily work, and to think aloud while using the prototype. Like in the first experiment, communication was reduced to a minimum, and we recorded our observations. After the participants completed their sketches, they had to define one instance of each occurring symbol. We stored the resulting symbol library and asked the participants to sketch another diagram of the same kind, re-using the previously defined symbols whenever possible.

At the end we asked debriefing questions about the usability and utility of the prototype, and how participants would like to further use and process the drawn sketches.

The interview, the briefing, and the debriefing together lasted for 30 to 40 minutes. Working with the tool took between 20 and 30 minutes.

2.6.2 Results

Qualitative feedback from the participants was very encouraging. All but one practitioner told us that they know situations where a polished version of the tool would be useful to them in practice (RQ 3). Two participants explicitly asked us to keep them up to

Table 2.2: Diagrams drawn on whiteboards and paper as reported by participants.

- Process models (block diagram with events and activities, BPM)
- Business models (clients, orders, storage dimensions)
- Mindmaps
- Flowcharts
- Architecture diagrams (layer cake)
- Use Case Diagrams
- System architecture models (dependencies between modules)
- Boxes and arrows
- Simplified activity diagrams (Boxes connected with signals)
- Transactions (systems and flows of information)
- Entity diagrams
- No particular diagram type, dependend on stakeholders
- Sequence diagrams

date regarding our research and provide a more advanced tool version.

When we asked participants whether they use paper and pencil, and whiteboards in their work, they all answered yes. Whiteboards and flip charts were mentioned to be more important than paper and pencil.

We also wanted to know from participants what kinds of diagrams they draw in practice on whiteboards and paper (RQ 3). Table 2.2 summarizes the answers. Participants answered that they do not just use certain diagram types, but they draw *"something similar to, but not quite a..."* or *"a simplified version of a..."* followed by the diagram type. Regarding UML models, use case diagrams

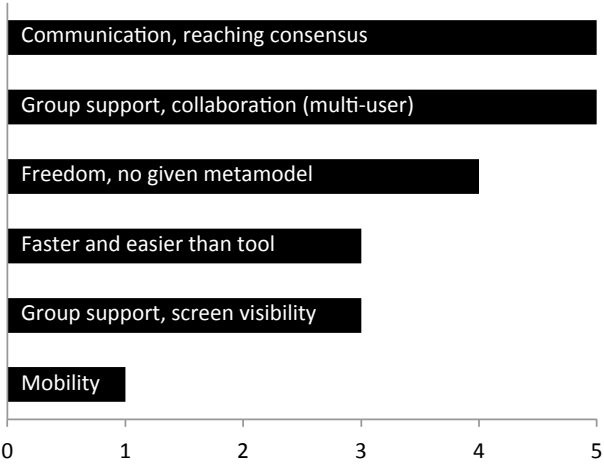


Figure 2.5: Most frequently stated reasons why paper or whiteboards are used.

were mentioned because they are *"something that stakeholders understand"*. Many participants said they draw diagrams to show some kinds of processes, e.g. business processes, and to show structures and relationships or transactions between stakeholders and systems, or between systems and subsystems. Two participants emphasized that they draw different kinds of diagrams dependent on their stakeholders' knowledge.

When we asked for the reason of using whiteboards or paper, the most frequent answer was that the large size of whiteboards and flip charts facilitates communication and reaching consensus (to reduce misunderstandings) in meetings and workshops (see Fig. 2.5). Participants also found it important to mention that

whiteboards allow for collaboration and encourage attendees to participate. The next reason for using physical media is that it supports free-form sketching. Participants mentioned that they find it faster and easier to draw on physical media compared to software tools. They stated that a large screen is preferred because many people can look at it at the same time.

We asked participants how the sketches drawn on physical media are re-used later on. Four participants said that they take a photograph and later re-create a model in another software tool using the photo as reference. Three participants stated that the content from the sketches is either communicated verbal, or a certain kind of documentation is created. Two participants usually take a photograph and put the picture directly into another document, where they add descriptions to it. Also, two participants told that in many cases the sketches only remain valid for some weeks, and are not used anymore afterwards (as they cannot be edited).

After the interview, participants used our tool to draw simple examples of their most used diagram types (RQ 3). Feedback about the usability and utility was similar to the one from the first experiment. *"A larger screen is needed"* was one of the most frequently given answers when we asked about their willingness to use an improved version of the tool in practice. Apart from this and the already mentioned usability issues, they were positive about a possible adoption in practice.

After participants have drawn diagram examples, we asked them what features exactly they would expect from an export function.

Here, the answers were quite diverse: some just want to be able to distribute it. Some want to have beautified versions of the sketches, while others are happy with the sketchy look. Most of the participants want to have it in an editable form (either beautified or not). One participant would like to export a list of the sketched entities into MS Excel. Another participant wants to work iteratively on the sketches, having the option to synchronize them back an forth between our tool and another software tool.

2.6.3 Findings

With this experiment, we investigated what practitioners usually sketch in practice, whether these sketches can be drawn with the FlexiSketch tool, and whether practitioners like our approach, i.e. would be willing to adopt our approach in practice. We also wanted to see whether they can classify the symbols they draw into different types.

We conclude that participants liked our approach as it suits the kinds of diagrams they draw (RQ 3): no standard diagrams, but something similar. Participants also frequently said that they draw UML-like diagrams, introducing their own notation.

Participants were also able to categorize the drawn symbols by assigning types (RQ 2). We were afraid that users might not be able to handle even this first step of lightweight metamodeling, but some participants actually wanted to do even more. For example, they asked how they could define different types of associations,

constraints, and how to declare a type as subtype of another one. However, other participants did not bother about metamodeling. This highlights the diversity of user skills which our approach should account for.

We probably have to trade the mobility of tablet computers against a larger screen (e.g. an electronic whiteboard) to gain a wider adoption in practice. Electronic whiteboards allow for multi-user input and thus facilitate collaboration, but they are far less widespread than mobile devices. Results suggest that the ability of a sketching tool to facilitate communication is important. This corresponds to findings from Ossher et al. [OBS+10].

When it comes to the question whether we should allow for an easier formalization of sketches by limiting the free-form sketching, participants agree that the free-form drawing is more important than formalization capabilities.

2.7 Threats to Validity

Construct Validity. We concluded that users are able to assign types to symbols. Depending on how we intend to transform informal sketches into semi-formal models, it might not be valid to state that users are able to provide relevant metadata if they are able to assign types. So we might be measuring what we mean to measure in special cases only.

Internal Validity. Participants voluntarily assigned types to symbols, as it was the only way to make copies of a symbol. This additional benefit can be seen as a motivating factor to define symbols, and therefore is a threat to internal validity.

Conclusion Validity. We did not try to measure a relationship between some treatment and some outcome, nor did we try to calculate statistical significance in our data. We were rather interested in assessing the feasibility of our approach and the utility of our tool.

External Validity. Our goal was to gather qualitative rather than quantitative data. However, the small sample size of our experiments is a threat to external validity. Many of the practitioners in our experiments have an academic background, and might therefore be more enthusiastic about our approach than software engineers in general. In the second experiment, participants decided to draw rather simple and high-level sketches because of time constraints and screen size limitations. Although these sketches contained key elements of drawings they use in real-world projects, this can be seen as a threat to external validity.

2.8 Related Work

Various mobile software tools for free-form sketching can be found online (e.g. Developer Whiteboard[2]). There are also modeling tools

[2]https://play.google.com/store/apps/details?id=com.agilaz.developerWhiteboard [last checked: 12/03/15]

for both general-purpose and SE-specific modeling. For example, DroidDia[3] allows to create different diagrams like flow charts, Venn diagrams, mind maps, and so on. It provides predefined symbols as well as basic geometric shapes. Although mobile computing is a recent trend in SE (e.g. [SGM10]), we are not aware of any existing work about mobile software tools that let users define and annotate their own diagram symbols. We fill this niche with our FlexiSketch approach.

Looking at desktop tools for sketch-based modeling support in SE, there are two threads of related approaches: (1) augmenting formal modeling tools with sketch recognition features, and (2) augmenting informal modeling tools (e.g. a tool mimicking a whiteboard) with features that allow a certain degree of formalization.

In thread 1, various approaches and tool prototypes have been developed that allow users to sketch diagrams (e.g. [CGH08, HD06]). The key idea is that sketch recognition algorithms will convert the produced sketches into semi-formal models. However, this also means that a user, when drawing a sketch, is still following the tool's predefined notations. Users therefore have to understand a specific modeling language in order to produce sketches that can be converted automatically. The sketch recognizer cannot interpret sketches that do not adhere to the language. Therefore, these approaches limit creativity and expressiveness. They also distract users from the actual modeling task [AD04] and can cause additional overhead due to sketch recognition errors.

[3] https://play.google.com/store/apps/details?id=com.alarex.gred [last checked: 12/05/15]

Approaches following thread 2 seem to be more promising. However, there are several challenges. For example, the set of possible sketches increases as soon as the user gains freedom in sketching. Thus, it gets more difficult for a tool to analyze and identify what the sketches actually mean. So far, only few researchers have tackled these issues and built tool prototypes that support users in drawing sketches independently of a specific modeling language. Such an example is the Calico prototype [MBD+10]. It provides mechanisms to structure sketches into different parts. Furthermore, a user can connect the parts with arrows. However, the user is not able to assign some meaning to the sketched symbols.

Other tools such as MetaEdit+ [KLR96] and MaramaSketch [GH07] include metamodeling editors. These editors allow to define a custom modeling language. The language definition is then used to compile a modeling tool for the defined language. However, these tools require to create the full language definition first and then users must strictly adhere to it, thus preventing any flexible sketching.

Some approaches do not require users to define symbols at the beginning. For example, BITKit [OBS+10] is a flexible modeling tool that focuses on combining the advantages of office and modeling tools. However, it does not include free-form sketching. The Electronic Cocktail Napkin [GD96] is a freehand drawing environment for conceptual design, and allows to incrementally transform sketches into schematic drawings. It does not focus on SE, but on architectural design (e.g. buildings). The open source project Sketch for Eclipse [SB10], currently under development, is

an API that provides sketching capabilities, a trainable gesture recognizer, and allows users to classify sketched elements. However, the project does not focus on user metamodeling. Since it is meant to be an API, it is not suited for end-users.

In summary, there is no existing solution that allows ad-hoc modeling and satisfactorily bridges the gap between free-form sketches and semi-formal models.

2.9 Conclusion and Future Work

In this paper we report on a tool-supported solution that combines free-form sketching with the ability to interactively annotate the sketches for an incremental transformation into semi-formal models. Software engineers using our approach have the possibility to evolve sketches into semi-formal models rather than re-modeling the information contained in the sketches in a software modeling tool. Our experiments provide first answers to our research questions: RQ 1: What are the differences between our tool and paper and pencil? User feedback indicated that sketching with FlexiSketch "feels" less natural and slower compared to paper and pencil, but additional functionality provided outweighs these limitations. RQ 2: Can users in general classify the elements they draw into types? Participants in our experiments had no problems doing so, therefore we answer this question with yes. RQ 3: Would users consider adopting our tool in practice? All but two participants were positive about adopting the tool in practice. We

cannot generalize this result for a larger user-base, so we answer with a tentative yes.

The major contributions of this paper are:

- A new, flexible process for sketching and the step-wise formalization of these sketches. The core of our approach is not specific to the SE domain, but can also be valuable for many other domains.
- A tool prototype that highlights the feasibility of our approach. Users can sketch first and assign meanings to the sketches on demand.
- Two initial usability and utility experiments showing that software engineers are able to sketch their favorite diagrams with our tool, and can annotate them.
- Insights about the use of informal sketching approaches in SE practice.

Recent technological trends and the growing market of tablet PCs provide the opportunity to come up with novel solutions supporting flexible, mobile, ad-hoc modeling in SE. Our work describes one possible approach how tool support for early SE phases may look like.

Although first evaluations indicate the usefulness of our approach, more research is needed to gain answers to open questions. Our future research will focus on the following:

Hardware for Sketching. Modern tablet computers usually come with a capacitive touch screen. Digital pens must have wide tips

in order to get properly recognized by the tablets. For some users, sketching with these pens feels unnatural. We need to investigate how we can make sketching feel more natural. Moreover, the small screen sizes of tables are an issue. Electronic whiteboards provide a large drawing space and facilitate collaboration, but limit end-user mobility and are far less pervasive than mobile devices. At the moment, a native Java version of FlexiSketch that runs on desktop machines and electronic whiteboards is planned, but our main focus remains on supporting SE activities with mobile devices.

End-user Lightweight Metamodeling. It is an open challenge how metamodeling can be made accessible to end-users [OBS+10]. Assigning types to symbols is only a first step towards a lightweight, end-user metamodeling approach. It is thus worthwhile to investigate how a more complete end-user lightweight metamodeling method could look like, and how it can be integrated into our approach. We want to explore how much information relevant for metamodeling is put into model sketches by users themselves, how much metamodeling is needed for exporting models into other software tools, and whether the tool could infer the missing information (semi-)automatically.

Field Studies. So far, we performed interviews and two controlled experiments. Once an improved version of our tool is ready for use in practice, we have to conduct case studies where our tool is used in the field during real-world projects. This allows to further assess the utility and adoption in practice of our approach. We want to explore in which situations our tool can be used, and how it will

be used. The studies will point out the benefits and limitations of our approach in more detail, and will enable us to refine it.

Acknowledgment

The authors would like to thank Sebastian Golaszewski, who made a big contribution to the FlexiSketch tool.

Chapter 3

FlexiSketch TEAM: Collaborative Sketching and Notation Creation on the Fly

Original publication:
FlexiSketch TEAM: Collaborative Sketching and Notation Creation on the Fly
D. Wüest, N. Seyff, and M. Glinz
International Conference on Software Engineering 2015

Abstract

When software engineers collaborate, they frequently use whiteboards or paper for sketching diagrams. This is fast and flexible, but the resulting diagrams cannot be interpreted by software modeling tools. We present FlexiSketch Team, a tool solution consisting

of a significantly extended version of our previous, single-user FlexiSketch tool for Android devices and a new desktop tool. Our solution for collaborative, model-based sketching of free-form diagrams allows users to define and re-use diagramming notations on the fly. Several users can work simultaneously on the same model sketch with multiple tablets. The desktop tool provides a shared view of the drawing canvas which can be projected onto an electronic whiteboard. Preliminary results from an exploratory study show that our tool motivates meeting participants to actively take part in sketching as well as defining ad-hoc notations.

Demo video: http://youtu.be/0kHjNfHLViM

3.1 Introduction

Software engineers frequently use whiteboards when they collaborate with each other or with project stakeholders in early project meetings. They create diagrammatic sketches for requirements elicitation, solution and problem design, viewpoint negotiation, and idea generation [CVDK07, MLPvdH14]. The resulting sketches often show notations that deliberately deviate from defined standards such as UML, are simple, and sometimes ambiguous [MLPvdH14, GD96]. On the one hand, creating such notations ad-hoc allows meeting participants to depict problems and ideas at any level of detail and in a form that can be understood by all participants. On the other hand, using non-standard (and potentially ambiguous) notations makes sketches hard to understand outside of the meeting context [DH07]. People who did not attend a meeting have to assume meanings for symbols. Even meeting participants might no longer be able to correctly interpret the sketches a few weeks later [CVDK07, WSG13a]. For re-using sketches during the software engineering process, engineers tend to either include pictures of them in documents, or manually build formal models from scratch, based on the sketches. The latter can be a time-intensive and error-prone task [WSG13a].

In our previous work, we developed FlexiSketch [WSG13a, WSG13b], a tablet-based tool for free-form sketching and the creation of arbitrary node-and-edge diagrams. FlexiSketch provides lightweight metamodeling functionality that allows the user to step-wise formalize the diagram sketches by assigning types and cardinality rules to sketched elements.

The main idea of our tool, which distinguishes it from other sketching tools, is that users can freely and seamlessly interleave sketching and metamodeling (i.e., defining a syntax for sketched symbols and links). This gives users the opportunity of both unconstrained sketching and modeling with a defined notation (that users may create themselves), including any combination of the two options.

So far, FlexiSketch has been an application for a single user. In this paper, we present FlexiSketch Team, a tool that supports synchronous, co-located, and multi-display collaboration for sketching and modeling in meetings. It consists of a significantly extended tool version for tablets and the new FlexiSketch Desktop for PCs that provides similar functionality and acts as a server. Our approach allows multiple users to work on the same sketch concurrently. Using their own tablets, participants can collaboratively work on a problem and also define a modeling notation ad hoc. This notation can be re-used in later software design sessions. The screen of the PC tool can be projected onto a wall or an electronic whiteboard and provides a shared view of the sketch canvas and the defined language constructs.

In the remainder of the paper, we first describe the core features of FlexiSketch (Sect. 3.2) and then present the new tool solution for collaborative work (Sect. 3.3), followed by preliminary evaluation results (Sect. 3.4). Sect. 3.5 discusses related work and Sect. 3.6 concludes.

3.2 FlexiSketch Basic

The single-user version of FlexiSketch is an Android tool[1] for model-based sketching (see Figure 3.1) [WSG13a]. Users can start by drawing free-form sketches. Whenever they lift the finger for a specified amount of time, the strokes are converted into a distinct symbol. Symbols can be selected and moved around. A context menu allows for further manipulation (e.g., resizing, adding text, deleting). A new feature allows users to import existing images into a sketch. Images behave like symbols and can be cropped. A link between two symbols is created by drawing a stroke from one symbol to another. Links have a context menu similar to symbols.

The context menu enables users to add semantics to symbols and links by assigning types and cardinality rules, and thereby define the vocabulary of a modeling language. All typed elements are copied into a type library. The type library is our construct that holds all types and the visual notation of a modeling language. A sketch recognition algorithm recognizes symbols that resemble user-defined types. By dragging a slider on the right side of the screen, users can reveal a container showing the entries of the type library. A drag&drop mechanism lets users re-use elements from the library. Type libraries can be changed at any time during the sketching process. The tool automatically builds a metamodel and infers cardinality rules for links, according to the current

[1] An early version of our tool is available in the Google Play store. A video demonstrating its features can be found at http://youtu.be/D06t0K5Otzw

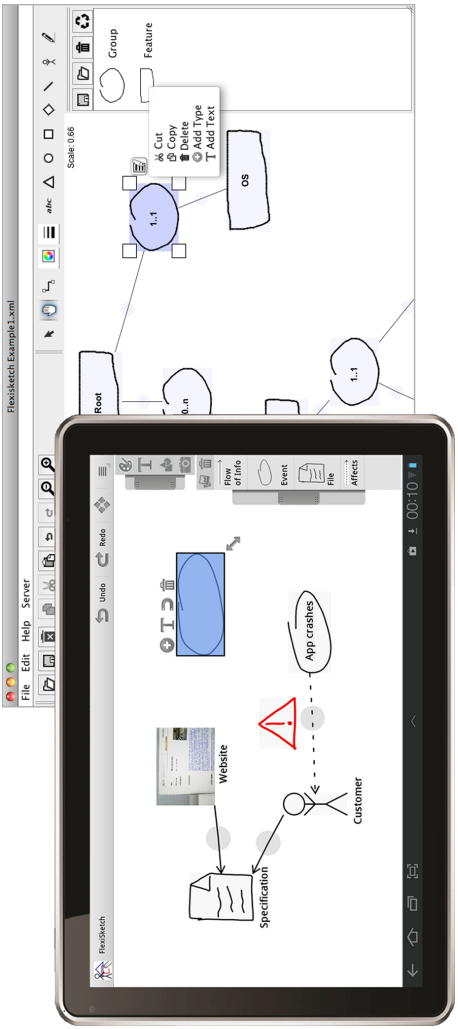


Figure 3.1: The Android version of FlexiSketch running on a tablet, and the desktop version running on a Mac (showing two different model sketches).

```
...
<Symbol>
        <type> File </type>
        <attributes>
                <labels> ... </labels>
                <min__occurrence> 0 </min__occurrence>
                <max__occurrence> N </max__occurrence>
        </attributes>
</Symbol>
<Link>
        <type> Flow of Info </type>
        <appearance> ... </appearance>
        <direction> UNIDIRECTIONAL </direction>
        <connections>
                <connection__1>
                        <from__element> Person </from__element>
                        <to__element> File </to__element>
                        <from_cardinalities>
                                <min> 0 </min> <max> 1 </max>
                        </from_cardinalities>
                        <to_cardinalities>
                                <min> 0 </min> <max> 1 </max>
                        </to_cardinalities>
                </connection__1>
        </connections>
</Link>
...
```

Figure 3.2: Metamodel excerpt from the tablet sketch in Figure 3.1.

sketch and the definitions in the library. Both the sketches and the metamodel can be exported in a structured form (e.g., xml files, see Figure 3.2).

We provide more details about our tool's metamodeling capabilities and a step-wise formalization of model sketches in [WSG13b].

3.3 FlexiSketch Team

The envisaged usage scenarios for our tool are requirements, design, and idea generation meetings in software development (see Section 3.1). We currently focus on co-located settings where communication between participants, apart from sketching, happens via natural language and gestures. Supporting geographically distributed synchronous collaboration is also possible, but not yet implemented.

Our tool provides a multi-screen setup where all meeting participants run the app on their tablets and have concurrent editing access to a synchronized canvas. The interface of FlexiSketch Desktop looks slightly different since it is adjusted to mouse & keyboard input (see Figure 3.1). It is also suited for electronic whiteboards: all actions can be performed by using the left mouse button (which is simulated by a touch), and the various parts of the interface (drawing palette, type library, etc.) can be moved and placed anywhere on the screen.

When participants meet, they connect their tablets to a server (a computer running FlexiSketch Desktop) via an ad-hoc wifi network, by e.g., using one of the Android tablets as mobile hotspot. Each participant has the option to scroll and zoom his/her own view to focus on different parts of the canvas. Optionally, the server is connected to a projector that provides a high-resolution overview of the sketch canvas and a list of all defined elements. This overview is automatically zoomed such that it always shows all drawings,

Figure 3.3: Meeting participants collaboratively create and discuss a model. The workspace is synchronized across the tablets and the electronic whiteboard.

but it can also be zoomed manually to focus on certain parts of the canvas and steer discussions (see Figure 3.3).

Drawings are synchronized as soon as strokes from a user get converted into a distinct element. The Android app adds a unique element and user ID, and sends the element to the server, which forwards it to all other tablets. For all subsequent manipulations of existing elements, only the IDs and the actual change (the delta) are communicated over the network. This helps to keep network traffic low and minimizes the time to propagate changes. The same applies to changes in the type library. The notation is synchronized between tablets and the libraries get immediately updated whenever a user assigns a type to an element or deletes a type from the library. The fact that sketches and notations are synchronized across tablets implies that all participants have

the meeting results on their personal tablets when they leave the meeting.

Users can not only sketch simultaneously within the same region of the canvas, but also define types of different elements concurrently. If two elements are created and the same type is assigned to both of them, the tool only generates one type entry, but stores both elements as alternative representations for that type. In contrast, when two users try to manipulate the same element at the same time, a non-optimistic locking mechanism prevents this. Only one user can access the context menu of an element at a time (to perform actions such as move, scale, delete, add text, assign a type, or define cardinalities). The main reason for the locking mechanism is to prevent inconsistent states of individual elements. Otherwise a user might, e.g., delete an element while another user is looking at the text entry popup for adding text to this element. For the manipulation of an element, the user first selects the element by tapping on it. The server locks the element on all other tablets where it no longer reacts to inputs and is shown with a red background (Figure 3.4). On the user's tablet, the element appears selected and shows its context menu. All manipulations performed by the user are immediately propagated. The element gets unlocked when the user deselects it (by successfully performing a manipulation or tapping the white part of the canvas). The locking mechanism also provides some user awareness in the form of visual cues showing what parts of the model are currently manipulated by other users.

FlexiSketch Team includes a share function that allows any user to push his/her current sketch and notation to the other users'

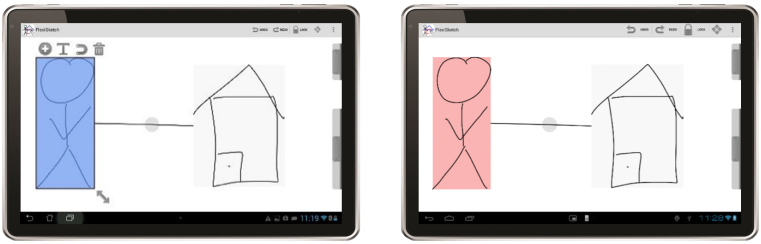


Figure 3.4: A symbol on the left tablet is selected and appears in blue. On all other tablets, the symbol appears in red and is locked.

tablets. This function can be used in a scenario where workshop participants prepare model sketches before the actual meeting, or when they individually sketch during the meeting. Participants can join an ongoing meeting and receive the current meeting artifacts with the push of a button. Furthermore, a user can disconnect her tablet to have a private workspace. When she is ready, she can re-connect and share her work with the other users.

3.4 Preliminary Evaluation Results

We conducted a qualitative study where we video recorded simulated workshops with three student teams and three SE practitioner teams from industry. Each team consisted of three co-located members. Here we concentrate on preliminary results from analyzing the practitioner teams. The teams were asked to choose a current SE related task or problem from their organization as a collaborative ideation and modeling task. We investigated how they

collaboratively sketched and defined notations on the fly when supported by our tool.

We found that all participants actively took part in the sessions and used the possibility to sketch simultaneously. In each team, the notations were defined by multiple participants. This happened incrementally during the sketching task, whenever they introduced a new element type, i.e., they interleaved sketching and metamodeling activities. All groups have chosen notations loosely based on existing standards by first agreeing on a known diagram type and then deliberately deviating from the notation standards (see Figure 3.5). Hence, discussions about semantics happened during the whole workshops. The type library with its drag&drop mechanism was heavily used. The possibility to re-use types motivated participants to define them, and led to diagrams with consistent notations. Regarding the usefulness of our tool, participants stated that defining types can also be seen as a form of documentation which helps to convey the meaning of elements and the sketch as a whole for later re-use. The practitioner groups reported that they enjoyed the flexibility of our tool compared to other software modeling tools, and that it is faster to share the created artifacts with others compared to classic whiteboards. They stated that they would prefer whiteboards for short-lived, small sketches, while they favor FlexiSketch for larger sketches, as well as for sketches that are, or will be, re-used.

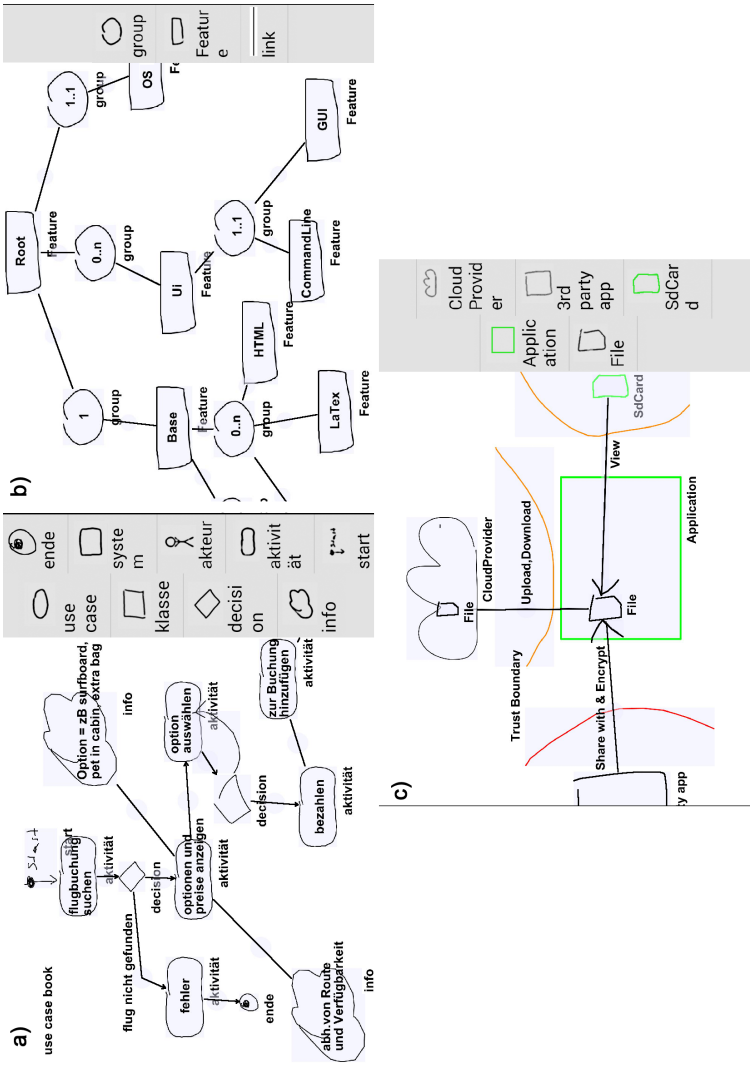


Figure 3.5: Result extracts from practitioner teams. The grey boxes show defined elements. Here, types are also displayed to the bottom right of each element.

3.5 Related Work

Collaborative sketching is an important method to foster creativity and discuss design ideas [GD96, VdL02]. Studies show that teams deliberately deviate from standard notations during early design meetings, e.g., Dekel and Herbsleb [DH07], and Ossher et al. [OJDB10]. A tool can only support such ad-hoc notations if it provides some kind of metamodeling mechanisms. While it was long believed that metamodeling should only be done by experts – and it has been shown that end-user metamodeling is indeed hard to achieve (e.g., [SCDLG12]) –, we argue that a form of lightweight metamodeling (or "just enough metamodeling") can be achieved in an end-user friendly way and is powerful enough for allowing the export of sketches as models.

The ability to collaboratively formalize arbitrary sketches of node-and-edge diagrams in a step-wise manner distinguishes FlexiSketch Team from similar work, which either provides sketch interfaces and recognition for predefined languages only, or allows for free-form sketching with little support for formalization. Examples for the former case include NetSketcher [BZS+11], and Scribble [SA13] which can inject sketching functionality into existing GEF based editors. CEL [LLO13] is a mobile tool that uses a minimal set of predefined element types, and exports models as skeleton source code. Examples for the latter are IdeaVis [GJPR10] and TEAM STORM [HHL+07]. Calico [MLPvdH14] and Idea Playground [PGS+12] allow for grouping and node-and-edge structures in free-form sketches. Some tools such as Knight [DHT00] and Tivoli

[PMMH93] include both informal and formal drawing modes, but no formalization of ad-hoc notations.

The Electronic Cocktail Napkin [GD96] provides functionality similar to the single-user FlexiSketch tool, but is more complex and needs programming/scripting knowledge. BitKIT also includes an incremental formalization approach [MOS+11], but focuses on specific data structures such as tables, while we focus on node-and-edge structures. Furthermore, FlexiSketch Team can be used to create a first draft of a custom modeling language (i.e., a DSL) by people without metamodeling expertise. Other metamodeling tools are hard to understand for non-experts [GHL+13] and/or do not allow seamless switching between modeling and metamodeling.

3.6 Conclusion

In this paper we presented our multi-screen, node-and-edge diagram sketching tool for software engineering. With FlexiSketch Team, participants can sketch simultaneously and use lightweight metamodeling mechanics to collaboratively define custom notations on the fly and step-wise formalize the drawings. The latter two features differentiate our approach from related work. Preliminary evaluation results indicate that our tool fosters interleaving of sketching and type-defining activities, and motivates all group members to take part in both activities. The groups managed to define consistent notations. In future work, we plan to extend

our approach with user awareness and communication features in order to support distributed collaboration. We plan to evaluate our tool in real software projects. We will investigate how sketches made with our tool are re-used and changed during projects, and gather feedback about the quality of sketches from the people who will actually re-use these artifacts.

Chapter 4

Semi-Automatic Generation of Metamodels from Model Sketches

Original publication:
Semi-automatic Generation of Metamodels from Model Sketches
D. Wüest, N. Seyff, and M. Glinz
Conference on Automated Software Engineering 2013

Abstract

Traditionally, metamodeling is an upfront activity performed by experts for defining modeling languages. Modeling tools then typically restrict modelers to using only constructs defined in the metamodel. This is inappropriate when users want to sketch graphical models without any restrictions and only later assign meanings to the

sketched elements. Upfront metamodeling also complicates the creation of domain-specific languages, as it requires experts with both domain and metamodeling expertise. In this paper we present a new approach that supports modelers in creating metamodels for diagrams they have sketched or are currently sketching. Metamodels are defined in a semi-automatic, interactive way by annotating diagram elements and automated model analysis. Our approach requires no metamodeling expertise and supports the co-evolution of models and meta-models.

4.1 Introduction

With the advent of model-driven engineering (MDE), models have become the main artifacts in a tool-supported, model-centric development process [Bro04]. Such an approach requires models to be machine-processable and transformable. Consequently, the corresponding modeling languages need to be defined precisely. Graphical modeling languages, on which we focus in this paper, are typically defined by a metamodel [AK03].

The standard way of using a modeling language is to define the language first, i.e., experts must create a metamodel for a new language before modelers can use the language for creating actual models [Kle08]. This allows the creation of powerful analysis and transformation tools required for MDE.

However, in the early phase of development, for eliciting, creating and sketching initial ideas, engineers want and need freedom in choosing notations adapted to their needs be it in the form of domain-specific languages (DSLs), by back-of-an-envelope style sketches, or both. Standard modeling languages such as UML are not well suited for that purpose. Instead, we need languages that can be flexibly defined and used in a way that they are well adapted for the specific problem at hand.

The traditional paradigm of upfront metamodeling breaks down here: Modelers want the flexibility to draw model elements regardless whether or not a pre-defined metamodel provides such

elements [OvdHS+10]. DSL designers want to draw sample models in a new DSL with full tool support before formally defining model elements in a metamodel.

At the same time, however, there is still a need for evolving such flexibly created models into a form that allows formal analysis and transformations. That means that metamodels must be created at some point. Metamodeling tools such as MetaEdit+ [TK09] and MetaBuilder [FHH00] provide some relief by making the task of formally defining a DSL easier and faster, but they still require upfront metamodeling. Today, modelers who need flexible modeling capabilities frequently use whiteboards for sketching [CVDK07, MBD+10]. This is done at the expense of later recreating the sketched models manually in a more formal modeling language in order to feed them into an MDE chain.

For really solving the flexible modeling problem, we need to *interleave modeling and metamodeling* activities and a tool that supports the *co-evolution of models and metamodels*. Combining sketching and metamodeling in a single tool is an approach not well studied so far. In our own previous work, we have developed the FlexiSketch approach [WSG12, WSG13a] which allows free interleaving of modeling and metamodeling tasks.

In this paper, we present how FlexiSketch step-wise and semi-automatically generates metamodels for model sketches, by inferring metamodel clues from existing model fragments and interactively eliciting missing metamodel information. Thereby, FlexiSketch enables modelers with no prior metamodeling expertise to

annotate sketched model elements with meanings and eventually produces a fitting metamodel.

The remainder of this paper is structured as follows. Section 4.2 provides the objectives of our research and information about FlexiSketch. Section 4.3 discusses FlexiSketchs metamodeling capabilities. Section 4.4 outlines first evaluation results. Section 4.5 presents related work. Section 4.6 concludes, discusses limitations and future work.

4.2 Modeling Languages and FlexiSketch

4.2.1 Focus and Objectives of our Work

We are interested in generating definitions of concrete and abstract syntax from a set of existing model sketches. Our goal is enabling engineers to create a language syntax definition for their early model sketches, such that

- these sketches can be formalized and re-used during the development process of a software project,
- engineers dont need metamodeling expertise,
- the tasks of modeling and metamodeling can be interleaved.

We restrict the scope of our work to graphical, node-and-edge style models. Typical examples for such diagrams are class diagrams, component diagrams or activity diagrams in UML. Also, graphical

DSLs typically fall into this category. This scope allows us to restrict the metamodel elements and structure we need to consider, omitting complicated structures such as deep inheritance trees that are hard to understand even for experts.

We focus on *collecting* metamodel information, both automatically by inference from existing model sketches and interactively in a tool-guided dialog with the engineer. Producing metamodels compatible with those of existing modeling tools is beyond the scope of our current work. However, we intend to generate metamodels that are sufficiently formal so that they can be transformed into a format understood by a commercial modeling tool.

4.2.2 FlexiSketch in a Nutshell

Our approach has been implemented prototypically in our FlexiSketch tool [WSG13a]. It is available for Android OS tablet devices and supports lightweight and flexible modeling. Having a mobile tool allows to use it in-situ in various contexts.

On start-up, FlexiSketch tries to mimic a whiteboard. Most of the screen is empty, inviting users to start sketching. User drawings are converted into elements that can be manipulated (e.g., moved, scaled, or deleted). FlexiSketch differentiates between symbols (nodes) and links (edges). Nodes may be drawn or consist of imported images. The tool allows assigning types to sketched elements by annotating them. These annotations provide the

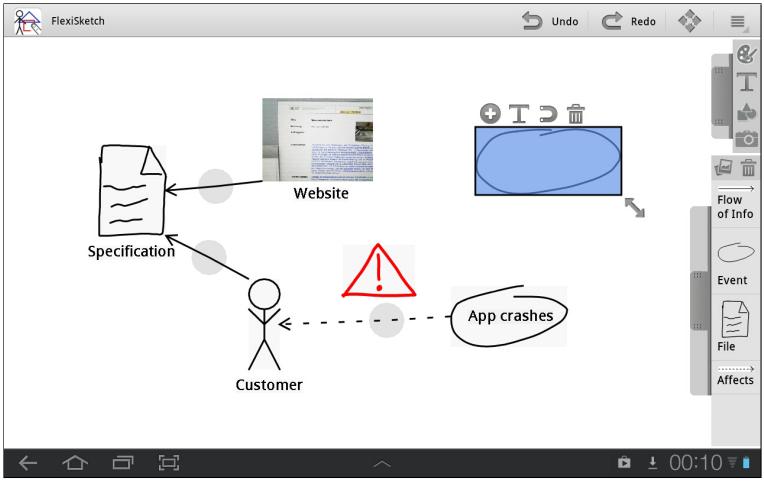


Figure 4.1: The FlexiSketch tool showing a user's sketch.

basic structure of a metamodel. Graphical representations of user-defined types appear in a *type library* on the right side of the screen. From there, users can create copies of their elements by dragging them onto the drawing canvas. Thus, the type library is a container for the user-defined concrete syntax. A sketch recognition algorithm processes the user-drawn symbols. If the user draws a symbol that looks similar to one from the type library, the tool asks in a small popup window whether it is the same symbol type. A more detailed description of FlexiSketch is given in [WSG13a].

Figure 4.1 shows a screenshot of the tool, showing a model sketch. The top right symbol is currently selected (indicated by a blue background and the visible context menu icons).

4.2.3 Using FlexiSketch – A Scenario

Engineers can use FlexiSketch to freely sketch their ideas as node-and-edge type models without any well-formedness constraints. If they decide to keep the resulting artifacts, FlexiSketch provides them with an easy, user-friendly way to add a metamodel to their sketches by annotating elements and answering questions asked by FlexiSketchs tool wizard. Once all the metamodel information has been collected, the model sketches and their metamodel(s) can be exported into an XML file. This file can then be transformed such that the models can be imported into other modeling tools, thus supporting an MDE approach. This encourages engineers to include their early sketches systematically into the software engineering process and avoids costly and risky re-modeling of information originally documented in sketches.

4.3 Metamodeling in FlexiSketch

In this Section, we give an overview on metamodel fundamentals in FlexiSketch, and then explain how we build metamodels based on inference and tool guidance. We also show how we minimize the versioning problem when associating metamodels with existing and new sketches.

4.3.1 The Metamodel Structure in FlexiSketch

In FlexiSketch, the user can create symbols, links, and annotations as elements on the drawing canvas. Symbols and links are

TypedElements, i.e., the user can define types for these elements. This creates a new *SymbolType* or *LinkType* class in the meta-model. Annotations are used to add informal notes to the sketched models. Furthermore, the meta-metamodel supports *Attributes* and *Containments*. Attributes can be used to add fields with type-value pairs to a symbol or link type. A containment gets created when a symbol is part of another symbol, i.e. it is drawn inside another symbol. The containment then stores the types of the symbols together with cardinalities defining how many symbols of a particular type may be contained in the symbol of the other type. Attributes and containments are not yet supported in the tool.

FlexiSketch does not store cardinalities directly for link types. Instead, it stores them for *ConnectionTypes*, which is a more flexible solution. While a link type is defined by just the type of the link itself, we uniquely identify a connection type as combination of the type of the link and the types of the two connected symbols. If the link is directed, we have a *start symbol* and an *end symbol*. A link type can have several connection types, i.e., when the same link type is used to connect different types of symbols. For example, a link type R may be used in one case to connect a symbol of type A with a symbol of type B, and in another case to connect a symbol of type A with a symbol of type C. Accordingly, the tool generates two connection types, one for *R(A, B)* and one for *R(A, C)*. The connection type for *R(A, B)* defines that R points from a symbol of type A to a symbol of type B. The cardinalities define (i) to how many type B symbols a single type A symbol may have

outgoing links of type R, and (ii) from how many type A symbols a single type B symbol may have incoming type R links.

4.3.2 Recognizing Elements on the Drawing Canvas

For the automated model analysis, we assume that the various elements in a model are already categorized into nodes and edges. This categorization is directly tied to how model sketching works in the tool. Whenever the user starts drawing and then stops for a certain amount of time, that drawing is converted into a distinct symbol which is always a node. Links (edges) can only be created by connecting two previously drawn symbols. For that, the user draws a stroke, starting inside one symbol and stopping inside another. The stroke is then automatically converted into a link between the symbols. Annotations are textboxes that are ignored for the metamodel creation, since they contain text related to a concrete model.

4.3.3 Inferred and User-Defined Symbol Types

Symbols on the drawing canvas can be selected. Upon selection, a context menu includes the option to assign a type (via text input) to the symbol. Each type appears in the type library together with its graphical representation, which is displayed on the right edge of the screen. From there, new instances of types can be

created by dragging and dropping them on the drawing canvas. This mechanism is an advantage of having a single environment for both modeling and metamodeling. It gives immediate feedback about all currently defined elements of the language. As the type library allows to re-use defined types, users can get motivated to assign types to elements, even if they do not intend to perform metamodeling [WSG13a].

Once some symbols are defined, the type of similar symbols can be inferred. A sketch recognition algorithm recognizes similar, yet untyped symbols. As recognition errors are inevitable, the tool does not automatically assign symbol types. Instead, it displays suggestions to the user in a small popup window. The user can either tap on suggestions or simply ignore them, as they disappear after a couple of seconds. As long as a symbol remains untyped, FlexiSketch internally uses a unique identifier for the symbol type. This is not shown to the user, but needed to distinguish untyped symbols from each other when connection cardinalities are inferred.

4.3.4 Inferred and User-Defined Link Types

Each link (edge) in the model represents a connection. Selecting links and assigning types to them works in the same way as for symbols. However, for the appearance of a link, the user has to choose from a predefined set of options (arrow, no arrow, solid line, dashed line, etc.). This allows the tool to guarantee a 1:1 correspondence between semantic constructs and graphical representations of links. If two links have the same appearance,

Flexisketch infers that they have the same type, thus prohibiting *symbol overload* [Moo10] for link types. As soon as the user assigns a type to a link, all links with the same appearance in the model automatically get the same type assigned. Conversely, FlexiSketch does not allow the user to assign the same type to two links having different appearances, thus preventing *symbol redundancy* [Moo10] for link types. The restriction of having a 1:1 mapping between link appearances and types also facilitates the inferring of connection cardinalities.

4.3.5 Inference of Connection Cardinalities

The user can define cardinalities for a connection type directly by selecting a link of that type on the drawing canvas and using the context menu to set the cardinalities. For connection types having no user-defined cardinalities, FlexiSketch automatically infers them, using a *closed world assumption*: the inference is based only on those links that have been modeled so far. This means that the tool infers very restrictive cardinalities in the beginning, starting with *1..1* when two symbols are connected with a link. When more links of the same type connect the same symbol with others, the cardinality rule is relaxed to, e.g., *1..4*. Thus, the tool never sets a cardinality to n; such generalizations must be done by the human user. Alternatively, the inference algorithm could be changed such that n is inferred whenever a cardinality is greater than *1*.

The tool infers cardinalities whenever one of the following events happens: (i) The sketched model is saved (the metamodel is

saved as well); (ii) The user wants to set the cardinalities of a connection type. Instead of presenting empty fields to the user, they are pre-filled with the inferred cardinalities (unless user-defined cardinalities already exist); (iii) The user locks the metamodel (see Sect. 4.3.7).

As described in Sect. 4.3.1, a connection type is defined as *R(A, B)*, where R is the type of the link, and A, B are the types of the connected symbols. To infer the cardinalities of a connection type, FlexiSketch looks at all occurrences of *R(A, B)* in the sketch. If a symbol of type A has less or more outgoing links to symbols of type B than defined by the current minimum and maximum outgoing cardinalities, the cardinalities are automatically adjusted accordingly. The same is done for defining the incoming cardinalities.

Cardinalities and connection types can also change when symbol types and/or link types are changed. A typical case is when a single link type is used to connect many untyped symbols. For each link, a connection type needs to be created, as each connected symbol potentially has a different type and different cardinalities. When the user then assigns the same type to several symbols, the according connection types are consolidated into one, and the cardinalities are updated. Rules can also get more restrictive when links are deleted. But cardinalities are never set more restrictively than the values already defined by the user. Deleting a link or a symbol from the drawing canvas can also alter the list of connection types. If a connection type has no more instances on the canvas, it gets deleted. However, link types and symbol types that are

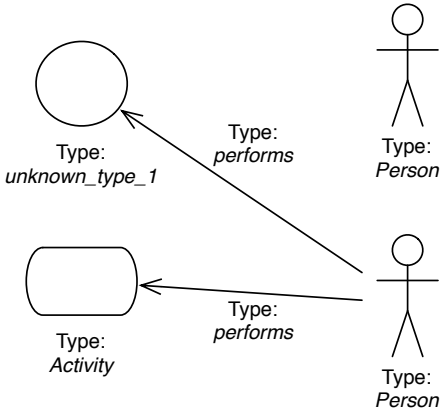


Figure 4.2: Inferring example.

defined by the user and visible in the type library are never deleted automatically.

Figure 4.2 shows an example. The tool manages two connection types *performs(person, activity)* and *performs(person, unknown_type_1)*. The type *unknown_type_1* indicates that the user has not assigned a type to this symbol. For *performs(person, activity)*, the tool detects that one symbol of type *person* is connected to at most one symbol of type *activity*, while the other *person* has no connection. It therefore infers the outgoing cardinalities *0..1*. The incoming cardinalities are *1..1*, as each symbol of type *activity* has exactly one incoming *performs* link from a *person*. The cardinalities for the other connection type are identical. If the user now assigns the type *activity* to the untyped symbol, the

two connection types are merged, because both now define the same connection: *performs(person, activity)*. Since a *person* is now connected to multiple *activities*, the *0..1* cardinality rule is relaxed to *0..2*. If the user deletes one of the *activity* symbols, FlexiSketch checks the rest of the sketch to see whether it has to set the *0..2* cardinalities back to *0..1*. This depends whether there is still another *person* symbol in the sketch that is connected to more than one *activity* symbol or not.

4.3.6 The Wizard – Interactive Guidance

In addition to adding types and setting cardinalities by using the context menu icons of sketched elements, FlexiSketch provides a wizard that helps modelers to supply missing metamodeling information. The wizard can be consulted on demand. It is passive in order not to distract the user from the modeling task. The wizard can be especially useful when it is called before saving the finished model sketch to ensure that no metamodel information is missing. Currently, the wizard consists of three steps: first it asks about types of unknown symbols, then about links, and finally about cardinalities for connection types. In each step, the wizard displays a separate page and question per element.

In the first and the second step, the wizard identifies untyped symbols and links respectively. When it detects one, it shows it to the user on the drawing canvas. If needed, the tool scrolls the canvas to make the element visible onscreen and then highlights it with a blue background. At the bottom of the screen, the wizard

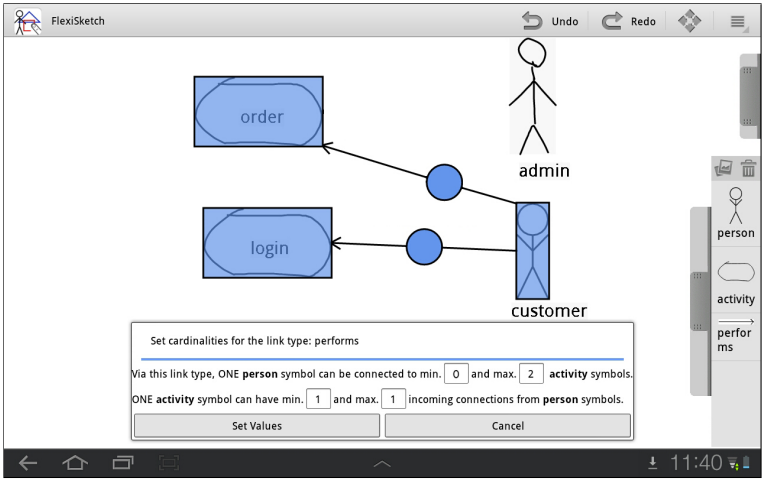


Figure 4.3: The wizard highlights an instance of a connection type and asks for the cardinalities.

asks the user to define the type. A definition can be skipped if the user does not regard the currently shown element to be relevant. As soon as the user assigns a type to a particular link, this type is automatically assigned to identical looking links.

In the third step, the wizard looks for connection types where at least one of the four cardinalities is not marked as *user-defined* (the state of a cardinality can be *inferred* or *user-defined*). When it finds one, it randomly picks an instance of it (a link) on the drawing canvas. It highlights the link and the connected symbols and provides the options to *set cardinalities*. Figures 4.3 and 4.4 show a screenshot of the wizard asking about cardinalities for a connection type from the inferring example in Sect. 4.3.5. The

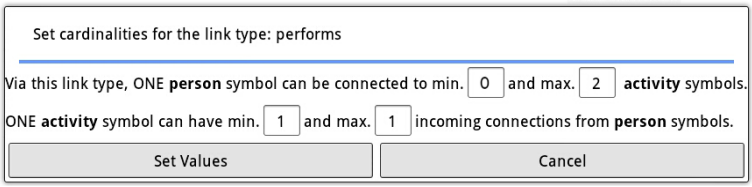


Figure 4.4: A close-up of the wizard window.

cardinality values in the fields (*0..2* and *1..1*) were inferred by FlexiSketch and are now presented to the user.

4.3.7 Storing Metamodels and the Lock Mechanism

The co-evolution of models and metamodels imposes challenges when it comes to storing the metamodels and how the versioning of metamodels should be handled [Wac07]. In our case, metamodels are undergoing an almost continuous evolution: as a user changes the model, this in turn might also change the corresponding metamodel. Earlier models that had the same underlying metamodel might no longer be compatible with the new metamodel version. We present two mechanisms to minimize the synchronization problem between multiple model sketches and the metamodel.

First, a metamodel is stored together with each sketched model. This ensures that each model conforms to at least one metamodel at any time. If two or more metamodels need to be merged into one, we can create a common metamodel automatically as long as the

result is a generalized metamodel, i.e., the merging can be achieved by only adding new meta-information and relaxing existing rules (e.g., changing cardinalities from *1..n* to *0..n)*. Such changes to a metamodel belong to the category of so-called *non-breaking changes* [CREP08]. Models conforming to one of the merged metamodels will also conform to the generalized metamodel. There are two more categories: *breaking changes which are resolvable*, and *breaking changes which are not resolvable*. Several researchers [CREP08, SCDLG12, RKPP09] present approaches for handling such metamodel changes.

Second, we introduce a lock mechanism. Metamodels can be saved independently from models. Once a metamodel is thought to be final, it can be locked. Other users can load a locked metamodel and start to sketch a model, but the lock disallows any changes to the metamodel. Therefore it will not be updated according to the model sketch. Instead, parts of the model that do not conform to the metamodel will be highlighted accordingly. This mechanism allows companies to leave metamodels unlocked as long as they are building DSLs. They can lock a metamodel to finalize their DSL, signaling modelers that they now have to adhere to the metamodel (unless companies want to unlock it again for improving the DSL).

Finally, metamodels can be saved and exported at any point in time. We illustrate this functionality with a small, sketched class diagram fragment as shown in Figure 4.5. The user has drawn three boxes and assigned the type *class* to them. She connected the box in the middle and the one on the right side with a link and

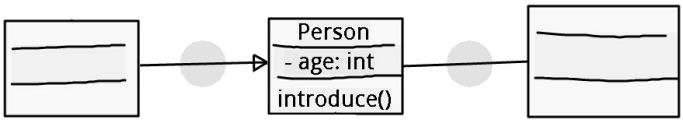


Figure 4.5: A minimalistic class diagram fragment.

defined it as *association*. Then she connected the third box with a link, changed its appearance to a line with an arrow head, and defined it as *inheritance*. The user then defined the cardinalities for the two connection types, and added some text to one of the boxes. Figure 4.6 shows an excerpt of the corresponding metamodel generated by FlexiSketch.

4.4 Initial Evaluation Results

We investigated to what extent FlexiSketch supports novice modelers in providing metamodel information for their model sketches. We conducted an experiment with 31 second term computer science students. The students had no prior metamodeling knowledge, and only little experience in modeling. After a tutorial in which the students learned about the tool functionalities, the students were assigned a use case modeling task. No introduction to metamodeling was given. However, the handouts stated that all model elements should be defined because the tool must be able to interpret the sketched diagrams. An online questionnaire completed the experiment.

```
...
 <Symbol>
   <type>Class</type>
   <attributes>
     <labels> ... </labes>
   </attributes>
 </Symbol>
 <Link>
   <type>Association</type>
   <appearance> ... </appearance>
   <direction>bidirectional</direction>
   <connections>
     <connection__1>
       <from__element>Class</from__element>
       <to__element>Class</to__element>
       <from cardinalities>
         <min>0</min> <max>-1</max>
       </from cardinalities>
       <to cardinalities>
         <min>0</min> <max>-1</max>
       </to cardinalities>
     </connection__1>
   </connections>
 </Link>
 <Link>
   <type>Inheritance</type>
   <appearance> ... </appearance>
   <direction>unidirectional</direction>
   <connections>
     <connection__1>
       <from__element>Class</from__element>
       <to__element>Class</to__element>
       <from cardinalities>
         <min>0</min> <max>-1</max>
       </from cardinalities>
       <to cardinalities>
         <min>0</min> <max>1</max>
       </to cardinalities>
     </connection__1>
   </connections>
 </Link>
...
```

Figure 4.6: Metamodel of the fully defined class diagram fragment from Figure 4.5.

Results show that the students were able to correctly define symbol and relationship types. In contrast, many students made mistakes in the cardinality definitions. The reason was that these students were thinking on the model level instead of the metamodel level, thus trying to assign cardinalities to individual relations instead of relationship types.

We performed this experiment with students to prove that FlexiSketch is easy to use and even modeling novices can generate metamodels with it. However, results suggest that users need at least some basic metamodeling knowledge in order to master metamodeling tasks that go beyond type assignment. Future studies will focus on requirements engineers, who are also the main target group of FlexiSketch.

4.5 Related Work

We identified related work about sketching and design in software engineering as well as metamodel inference. However, we are not aware of any work within the SE field that tries to combine a lightweight modeling approach (in this case model sketching) with a user-friendly metamodeling solution in a single tool. Most work about metamodeling focuses on the technical aspects, assuming that a metamodeling expert learns how to operate a formal modeling tool. The aspects of usability and user-friendliness are ignored.

The idea of bringing sketch interfaces and recognition into play to foster creativity and facilitate design tasks is not new [EHS69]. [BM08] presents a generic approach to generating diagram editors which support and analyze hand drawings. Several other researchers have incorporated a sketch interface into their semi-formal modeling tools, e.g., MaramaSketch [GH07], InkKit [PA04], and SketchREAD [AD04]. [JGHYLD09] gives a broad overview of similar approaches. While some of these approaches might allow users to alter the concrete notation, they only support predefined modeling languages. In contrast, the Calico tool [MBD+10] focuses on supporting an informal form of software design that heavily relies on sketching. It provides some means of structuring the sketches, but does not allow to formalize them. [DCJ11] and [VZJ11] discuss a step-wise formalization of models. New modeling languages can be created by linking artifacts of already existing languages.

Several research tackles metamodel creation from model examples, e.g., [GGLS11]. MARS [JMGB08] is a tool for reconstructing missing metamodels for a given set of models. Cuadrado et al. [SCDLG12] propose an interactive, bottom-up metamodeling approach similar to ours. But modeling and metamodeling cannot be performed in the same tool. Cho et al. [CGS12] focus on technical aspects of incremental and iterative metamodel definition by providing model examples. User interaction and tool-support are not discussed. Design guidelines for DSLs can be found in [KKP+09] and [POB00]. [CG11] and [SKT11] discuss design patterns for metamodels. Regarding user guidance, Qattous et al. [QGW10] demonstrate that defining metamodel constraints

with a by-example approach outperforms a form-based wizard approach.

4.6 Conclusions and Future Work

We have shown how FlexiSketch supports semi-automatic stepwise creation of metamodels, without the help of metamodeling experts. Following our research objectives given in Sect. 4.2.1, we presented strategies on how to gather information relevant for metamodel generation, using both automated model inferring and wizard-based user guidance. A key contribution of our work is our technical solution, which has been implemented in the FlexiSketch prototype. Experiment results highlight that modelers are able to provide relevant metamodel information when working with FlexiSketch.

FlexiSketch provides a lightweight and user-friendly approach to modeling and metamodeling. As we are mainly concerned about gathering basic metamodel information, we do not focus on building high-quality, sophisticated metamodels. Readers who are interested in the latter are referred to [CG11, POB00, KKP+09, SKT11].

In its current version, FlexiSketch does not support attributes (just textboxes as child elements of other elements), the nesting of symbols (containment), abstract classes, and inheritance in the metamodel. It also does not support any spatial information, e.g.,

there is no difference whether a symbol is placed to the right or to the left of another symbol. Feedback from practitioners about these missing properties suggests that containment and attributes are strongly needed, whereas inheritance and spatial information are less important features.

We are working on extending FlexiSketch with the missing features, especially attributes and containment, and plan to improve the wizard for better user-guidance. An export function will allow exporting the generated metamodels to MetaEdit+ [TK09]. We will also focus on the continuous use of FlexiSketch, which requires novel features regarding metamodel versioning. Other future work focuses on real-world case studies where practitioners use FlexiSketch within their daily work.

Chapter 5

An Investigation of Participant Behavior in Computer-Supported Collaborative Notation Creation and Sketching

Original publication:
Sketching and Notation Creation with FlexiSketch Team: Evaluating a New Means for Collaborative Requirements Elicitation
D. Wüest, N. Seyff, and M. Glinz
International Requirements Engineering Conference 2015

Abstract

Whiteboards and paper allow for any kind of notations and are easy to use. Requirements engineers love to use them in creative requirements elicitation and design sessions. However, the resulting

diagram sketches cannot be interpreted by software modeling tools. We have developed FlexiSketch as an alternative to whiteboards in previous work. It is a mobile tool for model-based sketching of free-form diagrams that allows the definition and re-use of diagramming notations on the fly. The latest version of the tool, called FlexiSketch Team, supports collaboration with multiple tablets and an electronic whiteboard, such that several users can work simultaneously on the same model sketch.

In this paper we present an exploratory study about how novice and experienced engineers sketch and define ad-hoc notations collaboratively in early requirements elicitation sessions when supported by our tool. Results show that participants incrementally build notations by defining language constructs the first time they use them. Participants considered the option to re-use defined constructs to be a big motivational factor for providing type definitions. They found our approach useful for longer sketching sessions and situations where sketches are re-used later on.

5.1 Introduction

Collaboration in Requirements Engineering (RE) often includes the usage of diagrammatic sketches to record and convey relevant information. Whiteboards and flip-charts are common tools used for brainstorming sessions to support requirements elicitation, design and idea generation [CVDK07, MBD+10]. Creating notations ad-hoc during such a session allows to describe ideas at various levels of detail, and often leads to simple, but also ambiguous sketches [GD96]. While engineers can choose notations that can be understood by all participating stakeholders, these notations typically deviate from standards such as UML [MBD+10, Tve02]. Therefore, the created sketches might be hard to understand for stakeholders who did not participate in the session and do not know the context and intentions behind the sketches [DH07]. These stakeholders have to assume meanings for symbols which might lead to wrong interpretations. Even for meeting participants themselves it can be challenging to correctly interpret sketches a few weeks later [CVDK07, GD96]. To re-use sketches during the RE process, engineers either take photographs and include them as non-editable files in other documents, or they manually build formal models from scratch based on the sketches, which can be a time-intensive task [CVDK07].

In previous work, we presented our first FlexiSketch prototype, a tablet-based tool for free-form sketching and the creation of node-and-edge diagrams [WSG13a, WSG13b]. In contrast to traditional sketching tools, it is possible to specify the sketched constructs

by, e.g., assigning types to them. Based on this information the tool infers a simple metamodel while the user is sketching. Being able to freely sketch models and at the same time creating a custom modeling notation on the fly allows to export and re-use the sketched models and metamodels in other modeling and metamodeling tools.

We have now extended our tool solution to support collaborative sketching and notation creation. FlexiSketch Team features synchronous, co-located, and multi-display collaboration. The technical details are described in [WSG15a]. Our approach allows multiple users to edit the same sketch concurrently using their own tablets. Users can also collaboratively define a modeling notation on the fly and re-use this notation in later RE sessions. They are free to choose notations that are comprehensible by all involved stakeholders, and can define a simple metamodel for it. To our knowledge, our approach is unique in the sense that it supports ad-hoc metamodeling in a collaborative sketching environment. This opens up interesting research opportunities regarding collaborative metamodeling.

The contribution of this paper is our conceptual solution for FlexiSketch Team, followed by an explorative study about how teams collaboratively define notations on the fly when supported by our prototype. The study includes two qualitative experiments: a laboratory experiment with students, and simulated workshop meetings with practitioners. In contrast to studies about collaborative sketching (e.g., [MBD+10, MLPvdH14, Tan91]), we wanted to investigate when and how teams define their notations while

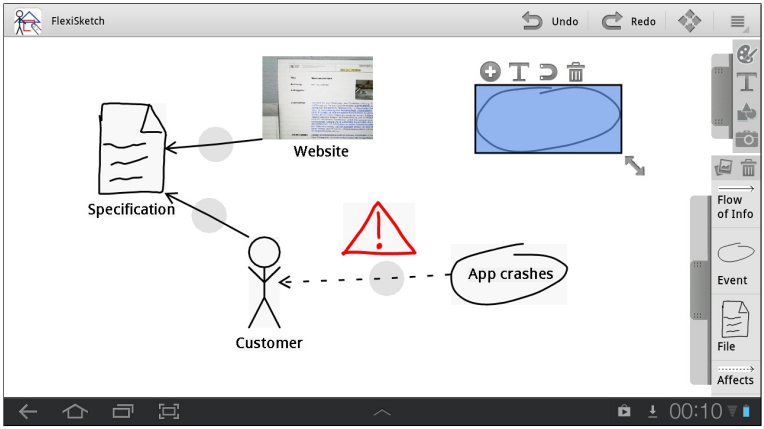


Figure 5.1: Screenshot of FlexiSketch showing the UI and a model sketch.

sketching, if the resulting notations are consistent, and whether all team members participate actively in defining the notations.

5.2 FlexiSketch

The single-user version of FlexiSketch is an Android tool for model-based sketching (see Figure 5.1) [WSG13a]. The main idea of our tool is that users can freely and seamlessly interleave sketching and metamodeling tasks (i.e., defining a syntax for sketched symbols and links). Users can sketch freely as well as draw models with a defined notation (that users may create themselves), including any combination of the two options.

Strokes from the user are converted into a distinct symbol when the user lifts the finger for a specified amount of time. If a user draws a stroke from one symbol to another, the stroke gets converted into a link between the symbols. Existing images can be imported into the sketch and behave like symbols. Each element on the screen can be selected and moved around, and a context menu provides additional editing features (e.g., resizing, adding text, deleting).

Users can also assign (arbitrary) types to elements, and thereby define the vocabulary of a modeling language. Our tool manages multiple type libraries. A type library contains a list of all user-defined types from the current sketch, together with their visual representations. Type libraries can be stored and loaded independently from sketches, and can be changed at any time. All types of a type library are shown at the right edge of the screen, from where users can re-use elements via drag&drop. Alternatively, a sketch recognition algorithm detects drawn symbols that resemble defined types. The tool infers cardinality rules for links and automatically builds a metamodel according to the sketch and type library definitions. More details about the metamodeling features and a step-wise formalization of model sketches are provided in [WSG13b].

5.3 FlexiSketch Team

In this section we describe our novel tool version that supports collaborative work[1]. Our envisaged usage scenario consists of brainstorming and design sessions in RE where the participating requirements engineers and other stakeholders collaborate in creating ideas, eliciting requirements, designing solutions, and negotiating viewpoints. We focus on co-located settings where communication between participants, apart from sketching, happens via natural language and gestures. As our tool allows for arbitrary node-and-edge diagrams, it is not only suited for RE, but also for a broader software engineering context. However, we focus on RE sessions as described above, because we believe that these are the sessions where informal diagram sketches are most frequent [CVDK07, GD96]: early in the software process, and when outside stakeholders (e.g., customers not knowing UML) are present.

5.3.1 Design Considerations

Analyzing our first tool version [WSG13a] and related work, we identified five key design issues (D) for the collaborative tool. These considerations also reflect selected design guidelines reported in research about computer supported collaborative work [Tan91, GG98, GG00, HLS+10].

[1] A demo video is available at http://youtu.be/0kHjNfHLViM

D1. *To foster active participation, all meeting participants should be able to concurrently sketch on the drawing canvas and define notations.* This allows to work in parallel and save time in writing down information [Tan91]. Further, participants should be able to choose where they stand or sit, while still having direct physical access to the workspace [HLS+10].

D2. The tool needs to *prevent conflicting inputs of participants.* It especially has to make sure that users can not concurrently change the defined notation in contradictory ways. This is closely related to coordinating the actions of participants as mentioned by Gutwin and Greenberg [GG00].

D3. The tool should *provide both shared and private views.* As it is harder to keep a shared focus when parallel work is supported [Tan91], a shared view can mitigate this problem and helps participants to be aware of each other's activities [GG98]. In addition, participants should have private views that they can manipulate. If these views are extended to private workspaces, users can take notes that are not shared with the group [HLS+10].

D4. *Results of a design session should be provided immediately to all participants.* Everyone should be able to leave the meeting in possession of the diagram sketches and the defined notations. Meetings exist "as part of a larger context of overarching activities" [HLS+10], and therefore some of the created information is likely to be re-used.

D5. The tool should *increase the awareness of each other's actions* by enabling participants to monitor each other [GG00]. Without

this support, it might be hard to tell who is doing what; especially when a user manipulates a diagram element via its context menu, and the resulting effect is perceived by others only when the manipulation already finished [GG98].

Due to time constraints, addressing issues D1 to D4 had to be given priority for our study about collaborative notation definition. A locking mechanism (see Sect. 5.3.2) that tackles D2 also partially addresses D5. This solution was sufficient for conducting our study. However, we will implement further awareness features in the future.

5.3.2 Technical Solution

Our novel tool version addresses design issue D1 by providing a multi-screen setup where all workshop participants have tablets and concurrent editing access to a synchronized canvas (see Figure 5.2).

Participants connect their tablets to a server via an ad-hoc wifi network. The server is a computer running FlexiSketch Desktop, a desktop version of our tool. This is a standalone version that is compatible with electronic whiteboards. Therefore, users can choose to work with tablets, on an e-whiteboard, or both. Alternatively, the server can be connected to a normal projector and shows an overview of the sketch canvas and a list of all defined elements. If not used actively, the desktop version automatically zooms its

Figure 5.2: Meeting participants collaboratively create and discuss a model sketch using multiple tablets and a synchronized drawing canvas.

view to always show the whole sketch, while each participant scrolls and zooms his/her own view on the tablet (D3).

As soon as strokes from a user get converted into a distinct element, or a user performed a manipulation on an existing element, this element gets synchronized across the tablets and the server (D1). Similarly, any changes to the type library are synchronized immediately. This also means that participants will leave the room with the meeting results on their personal tablets – without additional effort (D4).

With FlexiSketch Team, multiple users can sketch simultaneously within the same canvas region, and can define types of different elements concurrently. If the same type gets assigned to two different elements, the tool generates only one type entry, but stores both elements as alternative representations for that type.

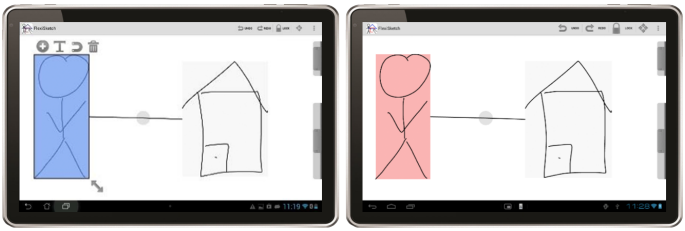


Figure 5.3: Left tablet: a symbol is selected and appears in blue. Right tablet: the symbol is locked and appears in red.

A non-optimistic locking mechanism [GM94] prevents inconsistent states of individual elements by prohibiting the concurrent editing of the same element: the context menu of an element is accessible by only one user at a time (D2). Otherwise a user could, e.g., delete an element while another user is in the middle of adding text or assigning a type to it. The server locks an element when a user selects it. On all other tablets, this element is then shown with a red background and does not react to user inputs (Figure 5.3). The user can de-select the element by tapping on any other part of the sketch canvas, whereupon the server unlocks the element. Using visual cues to show locked elements also provides some user awareness in the sense that users can see what model parts are currently edited by others. This is a first step towards D5.

A share function makes it possible to push the current sketch and notation from one tablet to the other tablets and the server. Therefore, meeting participants can, e.g., i) prepare different ideas before the actual meeting and then share and discuss them in the meeting, ii) disconnect their tablets during a meeting to have a

private workspace, and then re-connect to share their work and ideas (D3).

5.4 Study Goal and Method

The goal of our explorative study was to investigate how requirements engineers (both novices and experienced practitioners) collaborate in a workshop setting when supported with FlexiSketch Team. While the focus is on the collaborative definition of notations, analyzing the sketching behavior as well provides the necessary context. We refined our goal in three research questions:

Q1: How do collaborators sketch when they are provided with a collaboration tool that supports simultaneous sketching on multiple screens?

Q2: How do collaborators define and agree on a common modeling language and notation when they sketch?

Q3: What are the benefits and limitations of our tool-supported collaboration approach as perceived by the collaborators?

We conducted a laboratory experiment with graduate students (i.e., novices), followed by an observational study with practitioners in a simulated workshop setting. We included students for two reasons: (i) we wanted to test our approach with both novice modelers and experts, in order to assess how novices cope with our approach,

and to identify potential differences to experts. (ii) The student experiment also served as a test whether our tool prototype was good enough for showing it to industrial practitioners.

For the experiments, every participant received an Android tablet with the tool installed. The tablets had capacitive screens with sizes ranging from 9.4 to 10.1 inches. Participants could choose to use their fingers or a stylus. While we were not able to provide identical tablets for all participants, we believe that this reflects a real-world scenario where engineers bring their different, personal tablets to a meeting. We decided not to use an e-whiteboard for the study, as this allowed us to be more flexible where to perform our study (travel farther) and extend the amount of potential study participants. Instead, we used the desktop version of our tool to provide a shared view displaying the overview. The study was conducted in German. Quotations presented in this paper were translated to English.

5.4.1 Laboratory Experiment

The experiment was incorporated into an advanced requirements engineering course in a Swiss university, but we made clear that the students' performance in the experiment does not influence their grades. Eight graduate students in computer science were visiting the course, some of them having several years of industrial experience. One student already knew an old version of our tool because he had participated in an early usability study. The course size allowed us to form three groups. Group G1 consisted

of students S1 and S2, groups G2 and G3 had students S3-S5 and S6-S8 respectively. We found this to be a realistic group size for the kind of ad-hoc meetings that we want to support with our tool. The students already knew each other from solving group homework.

Our tool was introduced via a short training session before the experiment. We explained the main features of the tool and gave the students about five minutes to try out our tool in single-user mode. For the actual experiment, each group sat around a table. A computer, running the desktop version of our tool, was placed at each table and displayed the overview.

We gave the students two tasks and instructed them to solve these collaboratively within the groups, but we did not say how. The first task was to draw a use case (UC) diagram[2] for a web platform where students can share all kinds of documents. The second task was to create a user interface mockup (GUI) for the use case "sign up on the online portal". Both tasks were given in written form (in natural language) and included a prompt to be creative and depict as many ideas as possible, as well as to assign types to all elements on the sketch canvas. During the experiment, there was a supervisor assigned to each group who did not become active unless there was a technical problem. The students had ten minutes for each task, and the time was controlled by the supervisor.

[2] We predefined the problem and diagram types for the students in order to create a shared work context for each group, and because they were novice modelers – we did not want to risk overwhelming them with the creation of new modeling languages in the first-time evaluation of FlexiSketch Team.

At the end, each group had a discussion about the experiment for five to ten minutes, moderated by the supervisor. In addition, students were asked to fill out an online survey after the session[3]. Seven students filled out the survey.

The experiment data we collected and analyzed includes video recordings of each group, FlexiSketch log files listing user actions with timestamps, and participants' feedback from the discussion and survey.

5.4.2 Simulated Workshops

We organized three simulated workshop sessions with three requirements engineers per workshop. Again, we deemed this to be a realistic group size for ad-hoc meetings, and we wanted to keep a consistent group size over both experiments. Group G4 (practitioners P1-P3) consisted of practitioners from different companies in Switzerland, who are friends from their time at the university. The members of group G5 (practitioners P4-P6) work together in a university setting in Austria, but regularly deal with real-world problems from industrial partners. Practitioners P7-P9 from group G6 work together within a company in Austria. P4 and P8 are one hierarchy level above their co-workers. Other than this, we did not identify any power relationships. All practitioners except P2 did not know our tool before the workshop.

We introduced participants to the single-user version of our tool in a short training session (five to ten minutes) at the beginning. Then

[3] https://files.ifi.uzh.ch/rerg/flexisketch/StudentHandouts.pdf

we asked the participants to think about a current RE related task or problem from their organization. This was subsequently used as collaborative ideation and modeling task within the simulated workshop.

We then introduced the participants to the collaboration features. A projector was used to display the overview. We did not introduce the concept of a workshop moderator, because we wanted to see how participants organize themselves using our tool. We let the participants choose the seating themselves, in order not to influence their collaboration behavior.

The FlexiSketch meeting sessions, which were video recorded, were limited to 20 minutes. There was no interaction between the experiment supervisor and the practitioners during the sessions unless technical problems occurred. Semi-structured interviews concluded the sessions. The interviews also included the questions that we used in the student discussions and survey.

5.5 Analysis

One of the authors analyzed each video in two iterations. During the first iteration, he coded the editing behavior of each participant with a binary function (*1* if participant is currently touching the tablet, else *0*). Smoothing was applied by mapping the data to a function of discrete, two-seconds time steps in order to leave out fine-scale structures while keeping the important behavioral patterns. In the second iteration, the author coded the conversation

between the participants. Firstly, he created the coding scheme and conversation categories according to his experience from the first iteration, and then analyzed two videos (from student groups G2 and G3) to fine tune the scheme and categories. The outcomes were discussed with the other authors and research colleagues. After finalizing the coding scheme, the author processed the rest of the videos. For each participant and utterance, he coded whether the participant was speaking about the modeling language (*semantics*), the modeling task (*modeling*), tool-related subjects such as usability and specific features (*tool*), or topics unrelated to the task and tool (*other*). The semantics category includes utterances about the notation (e.g., "*What does this element mean?*", "*I'm going to draw and define an actor symbol*"). Modeling utterances are related to the domain model (e.g., "*We have a further actor, professor, who can also upload documents*"). Examples for tool utterances are "*Can symbols be rotated?*" and "*You need to hold down the finger to drag and drop*". Finally, an example for an unrelated utterance is "*You have nice drawing skills*". After the categories were created, we looked more closely at the semantics category to find out how participants communicated their type definitions (e.g., do they talk to their team members before or after creating a type? Do they discuss or just notify each other?).

Due to a software bug, we could not obtain complete logging data from all tablets. But where available, the tool logs were used for triangulating the video data. The data from the discussions, survey, and interviews was analyzed to gather data for Q3. Statements from discussions were grouped to find interesting patterns and recurring statements. We also looked for correlations between

survey answers, discussion statements, and participants' behavior during the experiment.

5.6 Results

5.6.1 Sketching and Collaboration Behavior

R1.1: Phases of simultaneous sketching happened in all groups. Figures 5.4 and 5.5 show when participants were talking and/or editing. All six groups revealed a working style where they had phases of silent, simultaneous editing and phases of discussions with and without editing. In the practitioner groups, all three group members were simultaneously editing during 13.5% (G4), 10.1% (G5), and 8.2% (G6) of the total session time. These values were higher for students with 51.5% (G1, the group of two), 20.1% (G2), and 23.3% (G3). This difference between practitioner and student groups correlates with the different amount of communication (see R1.2). Practitioner P4 (in G5) did not draw much, instead she helped by asking many explorative questions about the project and possible language constructs, e.g., "*Should we have different feature types or just one type called feature?*". We identified student S3 as a leader in G2. He talked the most and came up with many modeling ideas, while the other members concentrated more on actual sketching activities. No clear leader emerged in the other groups.

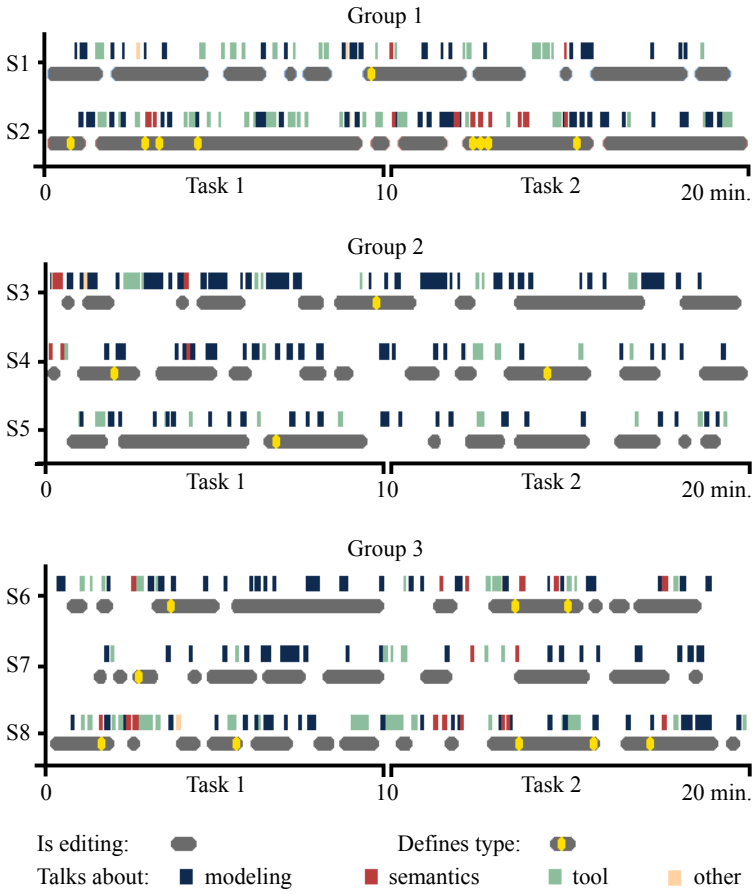


Figure 5.4: Phases of editing and discussions in student groups.

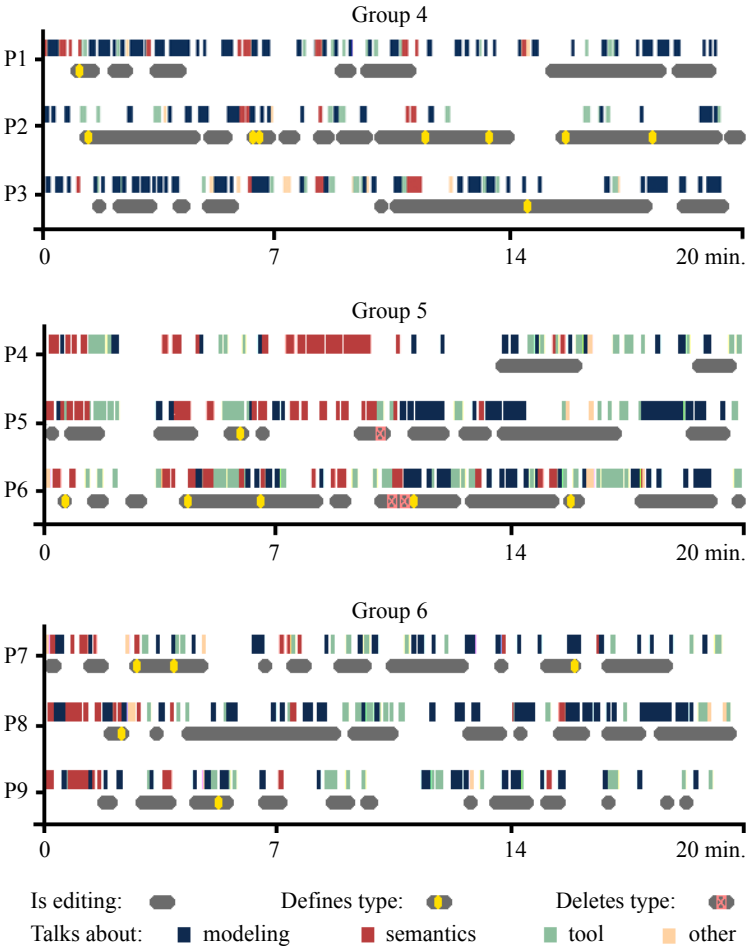


Figure 5.5: Phases of editing and discussions in practitioner groups.

R1.2: Practitioners communicated more than students. Practitioners were talking during 351 (G4), 415 (G5), and 314 (G6) of the discrete two-seconds time steps, which results in a mean of 12 talking minutes per group, while students talked during 203 (G1), 234 (G2), and 253 (G3) time steps, resulting in a mean of 7.7 talking minutes. Practitioners from all groups stated in the interview that there were no communication issues while working with our tool, and no group members disagreed. In contrast, students from G1 and G3 stated that their attention was drawn to the interaction with the tool. They believed that this reduced the amount of discussions they had, e.g., S2 said: "*Especially at the beginning we did not talk, each of us was concentrating on his own tablet*", and S3: "*Each of us drew something. We only discussed after noticing that two of us had sketched the same thing and we needed to agree about what to keep and what to delete*".

It rarely happened that a practitioner drew something without notifying the others about it. One exception happened in G4: At the beginning of the session, practitioners discussed every step before sketching something (e.g., P1: "*I'm going to draw a system boundary, okay?*"). Towards the end, communication regarding planned actions started to decrease: they were simultaneously sketching three different types of diagrams next to each other. They ensured consistency between diagrams by discussing key elements which were important for all diagrams, such as specific stakeholders and use cases.

No student group started with a brainstorming or extended discussion. Instead, communication happened rather "incremental":

multiple times during the session, they quickly mentioned ideas about what they could draw next and who will draw what parts, and then continued to draw silently.

R1.3: Participants tried to fit the whole sketch on their tablet screen. The diagram created by the practitioners in G6 fits on a tablet screen, while the diagrams from G4 and G5 clearly extended that size. In contrast, all student groups made their diagrams fit on a single tablet screen (in a way such that no scrolling or zooming of the canvas was needed). S5 from G2: "*We wanted to make sure that we always see the changes made by each other, and that no change happens outside of a tablet's current view*".

The usage of the big screen with the overview varied significantly between groups: G4 and G6 barely looked at it. P1: "*We used it once or twice*". In contrast, interview feedback and the video from G5 revealed that they used to look at the big screen when discussing the design and further steps. Similarly, five students stated in the survey that the big screen with the overview was useful (see Table 5.1).

R1.4: Participants peeked onto each other's tablet. All student group members were sitting close together, and all students took a look at others' tablets from time to time. Also, practitioners P1 and P2 in G4 and all three practitioners in G6 used to peek onto each other's tablet. P1: "*It helps to coordinate, to see what the other person is doing*".

Table 5.1: Student answers regarding FlexiSketch features on a Likert scale from "strongly disagree" to "strongly agree".

Activity / FlexiSketch feature	- -	-	o	+	++
Concurrent drawing was frequent	0	1	0	1	5
Modeling with the tool worked well	0	2	2	2	1
Many manipulation conflicts occurred	1	1	1	2	2
Lock mechanism was helpful	0	0	0	2	5
Lock is needed in same-place collab	0	1	2	3	1
Big screen was useful	0	1	1	1	4
I used drag&drop functionality	0	0	0	1	6
Drag&drop functionality was useful	0	0	0	0	7

5.6.2 Collaborative Notation Definition

R2.1: Notations were defined by multiple participants. The student groups defined a total of 9 (G1), 4 (G2), and 9 (G3) types, the practitioner groups defined a total of 9 (G4), 3 (G5), and 5 (G6) types. In all groups, type definitions were created by more than one participant (indicated by yellow dots in the sketching bars of Figures 5.4 and 5.5), with P4 being the only person who did not define any type.

For the student groups G1-G3 and practitioner group G4, the video analysis revealed that there were no discussions about the graphical representations of types, with one exception in G3 where S6 stated that he was about to declare a drawn symbol as *Use Case*. S8 intervened by asking him whether they should use a nice geometrical shape instead of the hand-drawn one, and S6 agreed.

In contrast, practitioner groups G5 and G6 briefly discussed in advance how the individual symbols representing the concepts should look like (see R2.4 for discussion details).

R2.2: Notations were defined incrementally during the whole sessions. All groups defined types whenever they introduced new elements in the diagram. Practitioner groups revealed a pattern where they discussed many semantics concerns in the early phase of the modeling task (especially G5 and G6, see Figure 5.5), followed by incremental discussions and ad-hoc notation definitions during the whole task.

R2.3: Participants based their notations on familiar concepts and symbols. Figure 5.6 shows defined types and extracts of the resulting diagrams from the practitioner groups[4].

All practitioner groups have chosen non-standard modeling notations loosely based on existing standards such as UML. Participants from G4 and G5 stated in the interview that they chose and agreed on concepts from languages that were familiar to everybody, and adapted them for use in their problem context. G4 used a notation which was very similar to UML, and most types were defined by P2 without discussions. The videos show that groups G5 and G6 started by discussing what types of diagrams they are going to draw, mentioning standard languages and important diagram elements. However, they then started to deviate from standards and introduced further concepts.

[4]The full diagrams can be found in high resolution at https://files.ifi.uzh.ch/rerg/flexisketch/TeamResults.pdf

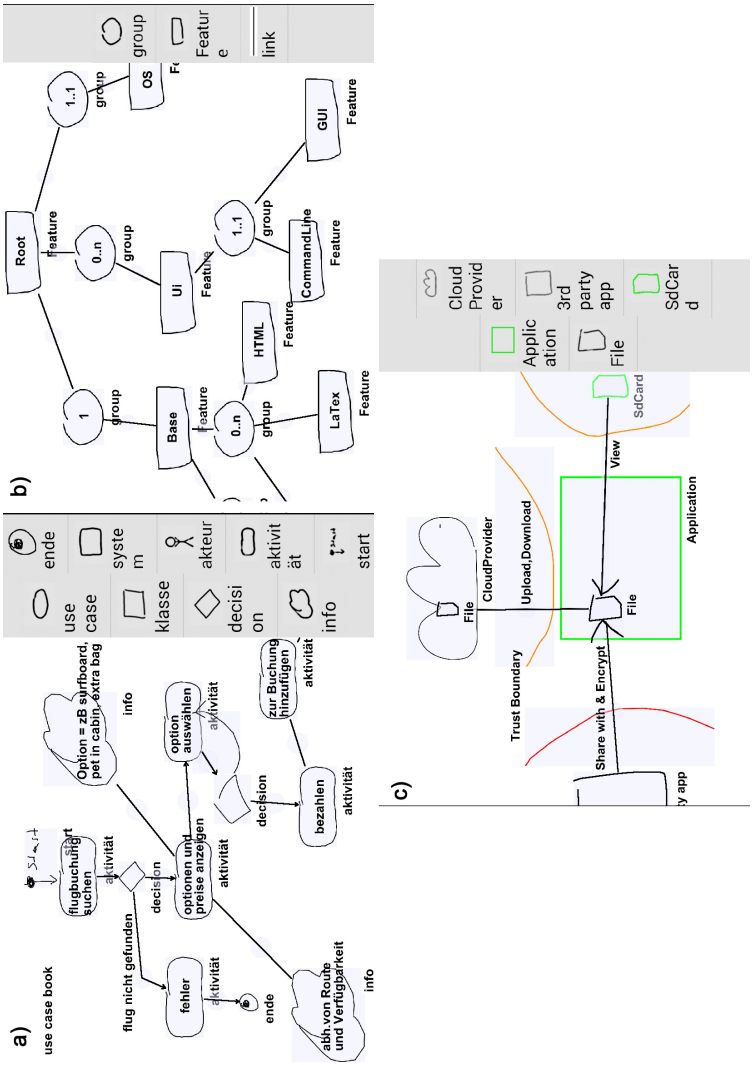


Figure 5.6: Extracts from the results of practitioner groups, a) G4, b) G5, c) G6. The grey boxes show the defined elements.

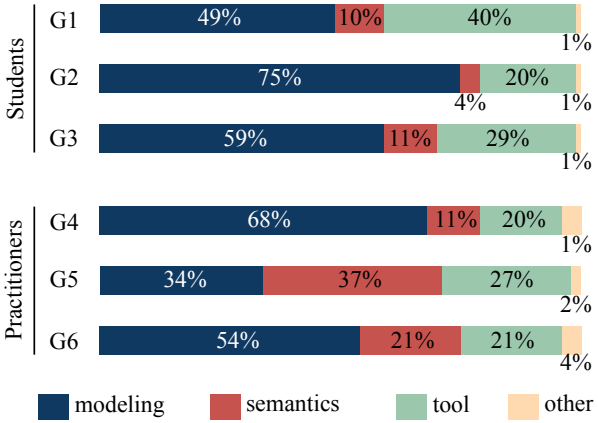


Figure 5.7: Talk category distribution in student and practitioner groups.

R2.4: Discussions about semantics depended on the chosen language constructs. In the practitioner groups, 11% to 37% of the total communication was devoted to semantics, depending on the group (Figure 5.7). G5 talked more about semantics than the concrete model, discussing a lot about how they can map their concerns to symbols. It was the only group that deleted some element types (Figure 5.5): in the middle of the modeling task, they discussed that three types have been defined at a too fine-grained level, and concluded to replace them by a more abstract type.

In contrast, group G4 discussed not much about semantics. P1 said: "*Borrowing most of the elements from UML allowed us to get a shared understanding of the symbols' meaning with little effort*".

Table 5.2: The amount of symbols, links, and defined types contained in each diagram from students (left) and practitioners (right).

		#symbols	#links	#types
G1	UC	12	7	5
	GUI	13	0	4
G2	UC	10	6	3
	GUI	3	0	1
G3	UC	7	5	4
	GUI	8	0	5

	#symbols	#links	#types
G4	20	16	9
G5	18	15	3
G6	9	3	5

Figure 5.7 reveals that student groups talked little about the meaning of elements, i.e., the semantics. P2 mentioned: "*There was no need to discuss because we were all familiar with the use case diagram notation needed for the first task*". Figure 5.4 shows that G1 and G3 communicated more about semantics in task two (user interface), while G2 did not talk about semantics. Indeed, results show that G2 almost completely neglected type definitions for task two (Table 5.2).

Figure 5.8 reveals how participants communicated their type definitions. In 40% of the cases (9 types), students talked about type assignments before they were actually doing them. In the other cases, they just informed each other by mentioning the symbol type, either while inputting the types or only afterwards (this behavior was especially noticeable during the GUI diagram task, e.g., S6: "*Radio button*", S8: "*Text field*"). In these cases, 75% of the communication (7 types) was initiated by a team member asking another one what he/she is doing or what the definition means.

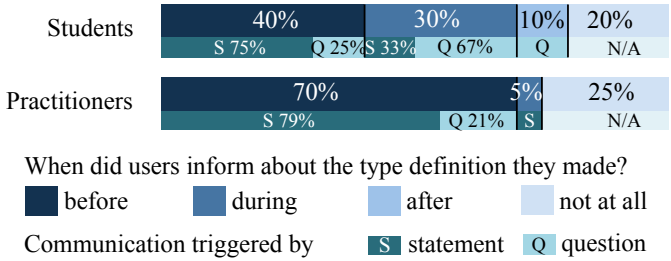


Figure 5.8: When participants talked about new types – before, during, or after defining them.

In contrast, practitioners discussed type definitions in advance in 70% of all cases (12 types). There were no questions related to a type definition after an element already got defined. In 20% of the cases, communication about a type definition was triggered in advance by a question (e.g., P8: "*Should we define a type named file?*"). The rest of the discussions were started by a statement.

Student groups did not discuss 20% of the type assignments at all. There was a similar high amount (25%) for practitioner groups. Especially G4, borrowing all element types from UML (except one type), defined four out of the nine types without any conversation.

R2.5: All groups created consistent notations. Table 5.2 shows the complexity of the resulting diagrams in terms of the number of defined types and elements drawn. The video analysis revealed that students from all groups always re-used existing types whenever possible. Therefore, all students within a group

used the same notation: all resulting UC and GUI diagrams that we received from students showed a 1:1 correspondence between symbols and meanings. In practitioner group G4, two symbols with different types cannot be distinguished by the sketch recognizer (*'system'* and *'class'*). P2, the person who defined the latter, told us in the interview: "*I wanted to sketch a better class symbol, but refrained from it because the session was about to end*". Apart from this case, the resulting diagrams showed no inconsistencies in the notation (all symbols can be distinguished by form and/or color). There was no evidence from the video analysis and the log files that inconsistencies happened during the sessions.

Regarding completeness, the type definitions were complete apart from some exceptions: G6 did not define the *'trust boundary'* symbol, and student group G2 did not define GUI elements. No group except G5 defined links (see Figure 5.6).

5.6.3 Perceived Benefits and Limitations

R3.1: The drag&drop mechanism was frequently used. Table 5.1 shows that all students liked the drag&drop functionality for defined elements. S2 said: "*As soon as you start to make bigger sketches, dragging elements [from the type library] onto the canvas is faster than drawing them by hand each time*". The video analysis and the resulting diagrams confirm that all but G6 heavily re-used the defined types by using the type library's drag&drop mechanism. P8 from G6 stated: "*The possibility to re-use defined types is a big motivation for defining them*". During the experiment, P8 said:

"*Can I rotate a symbol? ... No? ... In that case, I do not need to assign a type to this particular symbol*".

R3.2: Defined types can serve as documentation. Practitioners from all three groups said they liked having the sketches immediately available in digital form after a FlexiSketch session. Furthermore, P5 and P9 stated that types assigned to symbols also serve as some kind of documentation and contribute towards the comprehensibility of a sketch. P5 said: "*Due to the type definitions, I think I will have less effort in understanding a sketch when I look at it again after several weeks or months*".

R3.3: Participants liked FlexiSketch. Table 5.1 shows student answers to selected questions from the online survey. All groups reported in the interview that they liked the ability to draw simultaneously. P7: "*The tool makes it easy for multiple persons to draw on a small region of the canvas*". Practitioner P8 added: "*If you are, for example, ten people and have three tablets, I think this would be enough. You can circulate the tablets, and the others [who currently don't have a tablet] can look at the overview on the big screen*". Student S6 stated: "*The tool allows to sketch multiple ideas at the same time. Afterwards the team members can discuss the different ideas*". Student S4 said that the multi-screen setting takes some time to get used to: "*It depends on the setting. If everyone is at the same place, I'd prefer a big screen that allows multiple persons to sketch physically next to each other. But in a distributed setting, FlexiSketch comes in handy*".

Four students reported that many manipulation conflicts occurred (i.e., multiple students wanted to manipulate the same element

concurrently, which was prevented by the lock mechanism), while two students disagreed, and one was undecided. The video analysis confirms that concurrent manipulation did happen to different extents in the groups. Therefore, students stated in the survey that the lock mechanism is helpful. S5 added: "*Locked elements also provide visual clues about what the other group members are currently doing*".

Two students said in the discussion that they would have liked to have some kind of log, history, or color coding, in order to tell who has drawn what elements of the sketch. Two other students and two practitioners said that they would like to have an eraser function that allows them to erase only parts of symbols, as well as very small strokes that they made by mistake (currently, these strokes are hard to select and delete because of their small size).

R3.4: Groups prefer FlexiSketch for large and re-usable sketches. All groups said that they would prefer a classic whiteboard for coarse, short-lived and not too large sketches. With respect to size, the sketches created in the experiment were perceived to fall into this category. Only three students agreed that modeling with the tool worked well, while two were undecided and two were negative. They argued in the discussion that it is not worth dealing with some of the usability issues and having a less natural sketching feeling unless a sketch becomes bigger and exceeds the size from the experiment. Similarly, G4 and G5 stated that they would prefer our tool for larger sketches. P5: "*It will be easier to edit, store, and re-use them*". P1 said: "*FlexiSketch might unfold its advantages when officially introduced in a company and used for a prolonged time, over multiple workshops*".

Table 5.3: A summary of the results, grouped by research question.

R1.1: Phases of simultaneous sketching happened in all groups
R1.2: Practitioners communicated more than students
R1.3: Participants tried to fit the whole sketch on a tablet screen
R1.4: Participants peeked onto each other's tablet
R2.1: Notations were defined by multiple participants
R2.2: Notations were defined incrementally during the sessions
R2.3: Participants agreed on familiar concepts and symbols
R2.4: Semantics discussions depend on chosen constructs
R2.5: All groups created consistent notations
R3.1: The drag&drop function was frequently used
R3.2: Defined types can serve as documentation
R3.3: Participants liked FlexiSketch
R3.4: Groups prefer FlexiSketch for large and re-usable sketches

5.7 Discussion of Results and Design Implications

In this section, we discuss the results from the study (Table 5.3 provides a summary) and what they imply for the design of further collaborative sketching and notation definition tools.

Q1: How do collaborators sketch together? All participants took an active part in the sessions and used the possibility to sketch simultaneously (R1.1). Both students and practitioners took a look at others' tablets from time to time (R1.4), which can help in coordinating themselves (e.g., monitoring, assistance [GG00]). Loksa et al. [LMLvdH13] encountered the same phenomenon of "students peering onto the creator's tablet". R1.3 and R1.4 show

that user awareness is important in a setting where multiple small screens can be used for input. P1: "*When sketching collaboratively with a tool, you can just start to draw. But here, when defining types, you need to be more careful and coordinate*". A separate, big overview screen can reduce the problem to a certain degree. However, our results suggest that it is preferable to show the overview on the same screen a user is working on. This leads to a tradeoff regarding screen space [GG98] and asks for new solutions regarding the small size of mobile devices. Mobility is an important advantage of our tool. But for non-mobile tools, a shared screen and view could lead to smoother collaboration.

Some students had problems to manage both the cognitive and the social space [LTH12] at the same time (R1.2): they concentrated on the tool and did not communicate their actions well enough. This fits with a finding from Shih et al. [SNH+09] that users do not automatically "develop a sense of tolerance for lack of social awareness" in collocated sessions. However, studies suggest that it is possible to learn how to cope with a multi-space setting [LMLvdH13]. Indeed, we observed that practitioners did not have this problem and were able to coordinate their actions. This suggests that our tool needs additional awareness features to support less experienced users.

Q2: How do collaborators define and agree on a notation? Results R2.1 and R2.2 show that multiple participants in each team defined parts of the notation incrementally during the sketching task. All practitioner groups deliberately deviated from standard notations (R2.3). Dekel and Herbsleb found the same result [DH07].

Hence, discussions about semantics happened during the whole workshops (R2.4). While practitioners mostly communicated type definitions before they made them, students tended to perform the actions first, and talk about them afterwards. Compared to pure sketching environments, this collaboration style can lead to more confusion in our case because actions can also explicitly change the semantics. The student groups reported in the discussions that they noticed this and that they would probably focus more on their communication style if they receive a similar task in the future.

The drag&drop mechanism (that allows to re-use types) was heavily used and seems to have had a big effect on the notation definition behavior (R2.5, R3.1) and the consistency of diagrams. Firstly, it motivated participants to define symbol types right at the moment when they used them for the first time. Secondly, they re-used defined symbols whenever possible. In contrast to symbols, no group except G5 defined links. Possible reasons could be that link types cannot be dragged and dropped, and that FlexiSketch regards all links with the same appearance as being of the same (undefined) type, and therefore implicitly keeps a 1:1 mapping. Overall, our tool is an example of how a sketching tool can help to have consistent and unambiguous sketches at the end of a session if the users want this. A side-effect of the drag&drop functionality was that participants committed to notations early. Studies with physical media [DH07, OJDB10] show that the meanings of symbols are re-discussed and changed during design sketching, which rarely happened in our case (a possible explanation could be that our experiment consisted of a single, and relatively short, session).

Therefore, regarding creativity, it is an important design decision whether and in what form to include a type re-using mechanism in tools such as FlexiSketch. At least, users must perceive types to be easily changeable. A feature such as the typing mechanism can have both positive and negative effects: it can foster discussions about types and thus creativity, but it can also distract from the sketching task. S6 stated: "*The tool is great, but one also needs to think about a possible process. Maybe there could be a first meeting where participants only define the notation. And in the next meeting, participants can fully concentrate on the modeling task*".

Q3: How did collaborators perceive our approach? In general, our approach and tool features were very well perceived by participants (R3.1, R3.3), but they also mentioned minor usability issues and made clear that they would not use our tool in all situations (R3.3, R3.4). The practitioner groups reported to favor our tool for sketches that are, or will be, re-used (R3.4). Walny et al. [WHD+11] show that many sketches "undergo a variety of transitions" during software development. Sketches are re-used in different situations and contexts. Therefore, a flexible sketching tool should not just focus on supporting a single scenario (e.g., sketching in a workshop) or process step. Firstly, it should not impose a particular workflow on the user [OJDB10], and secondly, it should provide means for storing contextual information. Dekel and Herbsleb state that it is difficult to interpret artifacts without knowing the context in which they got created [DH07], which is especially true for (ambiguous) sketches. In that regard, practitioners stated that they also were motivated to provide type

definitions because they can serve as means for documenting the sketched diagrams (R3.2). At the same time, R3.1 shows that some users only provide this kind of lightweight metamodel information if they get an immediate benefit out of it. Furthermore, tools should capture the history (i.e., traces) of who did what, and when. Teams do not want to have to write down this information manually [DH07].

5.8 Threats to Validity

Conclusion validity. We conducted a qualitative study to get a first in-depth understanding how groups create ad-hoc notations. Quantitative studies are necessary in order to strengthen conclusion validity.

Internal validity. Participants were unfamiliar with the tool and its features for ad-hoc notation definition, which is a possible threat. To mitigate it, we gave an introduction to the tool. Yet, the desire of the participants to explore the new technology, as well as some minor usability issues, were potential distractions and could have influenced the collaboration task.

In a study like ours, participants might want to please the researchers by giving positive feedback. Therefore, we asked the students to fill out an online survey after the lecture, which allowed them to give feedback anonymously. The bias was mitigated for two practitioner groups by the fact that the second author was

not involved in conducting the experiment, and it was only him who knew a contact person from the groups G5 and G6.

Construct validity. We asked students to create specific types of diagrams, which can influence the amount of discussions needed about semantics, as well as minimize usability issues since we already knew that these diagrams can be built with our tool. However, the lack of micro-coordination that was revealed in student groups does not depend on a particular modeling notation. Furthermore, it was not a potential threat in practitioner groups, because they tackled real-world problems and freely chose notations.

External Validity. The limited number of students and practitioners who were involved in our evaluation activities, as well as the limited geographical distribution (Switzerland and Austria) is a known threat (convenience sampling according to proximity). However, we involved both novice and expert modelers with different backgrounds and skills to strengthen external validity. During the 20-minute sessions, we identified collaboration patterns that confirm the usefulness of our FlexiSketch approach. The generalizability to longer sessions has yet to be verified.

5.9 Related Work

In requirements and software engineering, collaboration is often researched in the context of design [LMLvdH13, BGCB10] and user interface creation [LMLvdH13, JA07, SBV12]. Collaborative sketching is an important method to foster creativity and

discuss design ideas [CGH08, GD96, VdL02]. To better understand the creative activities in software design, researchers studied how and why engineers use physical media (paper, whiteboards) specifically for software design, e.g., [CVDK07, GD96], which motivated us to conduct research about more flexible modeling tools. Other researchers are more focused on understanding the behavior and low-level collaboration patterns of participants when working with physical media, e.g., [Tan91, GG00]. The findings resulted in design guidelines for software tools that support collaborative work [Tan91, GG98, GG00, HLS+10]. We connected the requirements for FlexiSketch with these guidelines to come up with a collaborative version of our tool. There are many software tools that support collaborative sketching and design work (e.g., Calico [MBD+10], The NiCE Discussion Room [HLS+10]). Settings with such tools can result in different collaboration behavior compared to physical media (e.g., because workspace awareness differs). Therefore, the influence of software tools on collaboration and sketching behavior has been studied in e.g., [MBD+10, MLPvdH14, HLS+10, LW08].

While we also looked at collaborative sketching behavior when using FlexiSketch, the main focus of our study was to investigate how requirements engineers collaboratively define notations. Related work on this subject is still scarce. One reason is that, from a metamodeling perspective, it was long believed that metamodeling should only be done by metamodeling experts [Kle08]. Indeed, it has been shown that end-user metamodeling is hard to achieve [QGW10, SCDLG12]. In contrast, we concentrate on lightweight metamodeling (or "just enough metamodeling") for

creating ad-hoc notations in an end-user friendly way (e.g., for requirements engineers and domain experts). This scenario leads to the question how teams decide and agree on notations. Dekel and Herbsleb [DH07] performed an observational study to find out what kind of notations are used in object-oriented design, and how they evolve during sessions. Ossher et al. [OJDB10] investigated notations used in software design sessions to conclude whether their flexible modeling approach can provide appropriate support. Both studies used physical media in the sessions. In contrast, our study investigates how non-expert metamodelers choose and define notations when using a flexible software tool.

Compared to other studies such as e.g., [SNH+09], we do not primarily focus on the quality of the results, but we are interested in evaluating the behavior of the participants in terms of micro-coordination [LTH12] during notation creation.

5.10 Conclusions and Future Work

In this work we presented a qualitative study about how requirements engineers sketch and define ad-hoc notations collaboratively when supported by a flexible modeling tool. Our multi-screen, node-and-edge diagram sketching tool allows users to define custom notations on the fly by assigning types to elements. The qualitative study indicates that the tool fosters interleaving of sketching and type-defining activities, and motivates all group members to perform both activities. Users managed to define

consistent notations for their sketches collaboratively and reached a common understanding of the respective notations.

Results such as R1.3 and R1.4 suggest that having additional awareness features in the tool (knowing what the other users are doing) would be beneficial. In our future work, we plan to improve FlexiSketch according to these results. We also plan to perform longitudinal evaluations in industrial software projects, and investigate how sketches made with our tool are re-used and changed during projects. This will allow us to gather feedback about the quality of sketches from people who will actually have to re-use these artifacts.

Chapter 6

Lightweight End-User Metamodeling with FlexiSketch

Original publication:
FlexiSketch: A Lightweight Sketching and Metamodeling Approach for End-Users
D. Wüest, N. Seyff, and M. Glinz
Submitted to SoSyM for publication

Abstract

Engineers commonly use paper and whiteboards to sketch and discuss ideas in early phases of requirements elicitation and software modeling. These physical media foster creativity because they are quick to use and do not restrict in any way the form in which content can be drawn. If the sketched information needs to be re-used later on, however, engineers have to spend extra effort for

preserving the information: they could take a photograph and accept the fact that modeling software cannot process photographs, or they could manually re-create the sketched information in a formal modeling tool. While saving information in a machine-readable way comes for free with software modeling tools, they typically anticipate the use of specific, predefined modeling languages and therefore hamper creativity.

In this article, we describe a flexible tool-supported modeling approach that augments a sketching environment with lightweight metamodeling capabilities. Users can create their own modeling languages by defining sketched constructs on demand, and export model sketches as semi-formal models. We evaluated our approach with novice modelers and experienced practitioners in two studies to find out if they manage to use our lightweight metamodeling mechanisms correctly, and how they build notations collaboratively. Results show that experienced modelers adopt our approach quickly, while novices have difficulties to distinguish between the model and metamodel levels and would benefit from additional guidance and user awareness features.

6.1 Introduction

Despite all technological advances, physical media such as whiteboards, flip charts and paper still play an important role in software projects [CVDK07, Goe95, VdL02, Tve02]. Engineers particularly use them to sketch and discuss new ideas. These creative activities may happen anytime in a project, but are especially important for early project phases where people discuss requirements or early solution ideas [CVDK07, GD96, MBD+10]. In a creative process, engineers need to sketch ideas in any form and on different levels of detail, sometimes using elements of a modeling language, sometimes inventing notations on the fly [MLPvdH14]. Frequently, notations will be chosen such that involved business stakeholders (e.g., customers, domain experts) can understand them without explicit training [DCC+12]. Ambiguity is accepted as an important characteristic of creativity and idea generation [GD96]. However, the resulting sketches are difficult to re-use and to integrate into the documentation of the emerging system [BGCB10]. Documenting them as uninterpreted photographs hampers later re-use and understanding, as the interpretations that the creators had in mind are lost. Re-creating the sketched information as models in an established modeling language could preserve intended meanings, but is a tedious and error-prone manual translation process [CVDK07, OBS+10]. Further, as more time passes before this translation happens, it gets more difficult to remember the original intentions and to perform an accurate translation. Therefore, there is a need for method and tool support for creating sketches on suitable media and documenting them such

that the sketched information can be re-used and its interpretation is preserved [OBS+10]. This need becomes even more urgent with the increasing popularity of model-driven engineering, a software development methodology in which models play a central role in the engineering process.

Although many sketches represent diagrams in some modeling language or another, classic software modeling tools are not suited for supporting this kind of creative early modeling. This is due to the fact that modeling tools restrict modelers to the use of a predefined modeling language with strict rules for syntax and semantics [PA04]. It enables the tools to provide comprehensive support for creating and verifying models, leading to precise documentation with little or no ambiguity (or even enabling the automatic generation of source code). However, this advantage comes at the expense of not allowing engineers to create free-form sketches or models not adhering to the pre-defined language syntax. These restrictions stifle creativity [CVDK07] and hinder the flow of thoughts.

In our previous work, we have proposed an approach and developed an associated tool [WSG13a, WSG13b, WSG15a], which aim at supporting creative sketching without the disadvantages of cumbersome re-use and documentation of the sketched information as mentioned above. Our tool provides a sketching interface and enables users to perform lightweight metamodeling by annotating the elements they have sketched with meanings. A simple metamodel gets created semi-automatically on the fly by processing this information and by inferring cardinality rules. As a result, each sketch is stored together with a simple metamodel. How accurate

and detailed this metamodel is depends (to a certain degree) on the amount of user annotations. Users are free to decide how much metamodeling they want to perform, according to whether and how they want to re-use the sketches. Thus, our approach provides the flexibility of paper or whiteboards, but at the same time facilitates the integration of model sketches into the overall system design process.

Our tool comes in two versions: FlexiSketch uses tablets as sketching media and supports inexpensive, mobile sketching at any time and in any place. FlexiSketch Desktop runs on electronic whiteboards and can provide a big screen for co-located meetings. Multiple tablets can be connected to the desktop version over Wi-Fi, which allows users to collaborate and simultaneously work in the same workspace with multiple screens.

In this article, we report on two studies we conducted to evaluate our approach. Central to our approach is the interweaving of sketching and lightweight metamodeling activities, which was also the focus of the studies. Our main goal was to see how well modelers can use the metamodeling features of our approach, and how they define modeling languages in a collaborative setting. In contrast, measuring the quality of the created modeling languages or the generated metamodels was not within the scope of the studies[1]. We particularly answer the following two research questions:

[1]Since in our case the motivation for performing ("just enough") metamodeling is to formalize sketches rather than defining high quality modeling languages, it does not make sense to compare the quality of the created metamodels with those built by metamodeling experts at this stage. However, a future study about metamodel quality can help to improve the generated metamodels and thus enhance the reusability of FlexiSketch artifacts.

RQ 1: *What patterns of sketching and language definition emerge when modelers collaboratively define lightweight modeling languages with our approach?* We were interested to find out how small groups behave when they define a lightweight modeling language. RQ 1 includes sub-questions such as: are there recurrent patterns between the groups? Are there moments of simultaneous sketching? How many group members define a part of the modeling language? When do they perform these definitions? To answer these questions, we performed a study consisting of simulated workshops with small groups of students (novice modelers) and practitioners (expert modelers).

RQ 2: *Can novice and expert modelers define lightweight modeling languages correctly and completely with our approach?* We were particularly interested to find out whether novice modelers manage to define all of their model constructs with our approach, and if their definitions make sense. In other words: if a metamodeling expert would define the same model constructs (limited to the definitions that are possible to create with our approach), and we would take these definitions as ground truth, how well would the solutions from the novice modelers match this ground truth? If the solutions would match the ground truth reasonably well, this would mean that our approach allows users to actually create lightweight metamodels without the help of metamodeling experts. To answer this question, we performed a study consisting of a quantitative experiment with more than 100 students in computing. We complemented the study with a qualitative experiment with experienced practitioners, in order to see whether they find our

approach useful for the kinds of customized modeling languages that they use in practice.

This article is an extension of an existing conference paper [WSG15b]. The conference paper reports on the simulated workshops and discusses RQ 1 for small groups of students and practitioners. The new contributions of this extension are:

- the results of the second study consisting of a quantitative experiment with students and a qualitative experiment with practitioners to answer RQ 2;
- a proof-of-concept export module for our tool that enables the export of created model sketches and metamodels to MetaEdit+ (a commercially available metamodeling tool) [KLR96]; and
- a more detailed, complete overview of our approach.

The remainder of this article is structured as follows. Section 6.2 describes our tool-supported approach. Section 6.3 reports on the first study consisting of the simulated workshops for answering RQ 1. Section 6.4 reports on the second study consisting of the quantitative and qualitative experiments for answering RQ 2. Section 6.5 summarizes and discusses the findings from both studies. Section 6.6 presents an overview of related work. Section 6.7 provides conclusions and future work.

6.2 A Tool-Supported Approach for Flexible Modeling

In this section, we summarize the FlexiSketch approach, present our tool, and briefly describe the intended usage scenarios.

6.2.1 Core Ideas

The main ideas of FlexiSketch are the representation of sketches as node-and-edge diagrams and the interleaving of modeling activities (i.e., drawing elements) and lightweight metamodeling (by annotating drawn elements) [WSG13a, WSG13b]. This means that FlexiSketch users can perform modeling and metamodeling activities in any order. For example, they can first draw a model and then annotate it, or alternate between modeling and annotating, or create a metamodel for a domain specific modeling language (DSML) first and then draw a model using this DSML. A sketch in FlexiSketch consists of a set of individual elements, distinguished as symbols (nodes) and links (edges). This provides a basic structure upon which the metamodeling capabilities of FlexiSketch build. Metamodeling happens by assigning types to nodes and links, and defining link cardinalities.

6.2.2 Sketching and Editing

Figure 6.1 shows screenshots of the mobile and desktop versions of FlexiSketch. The mobile version of FlexiSketch runs on Android

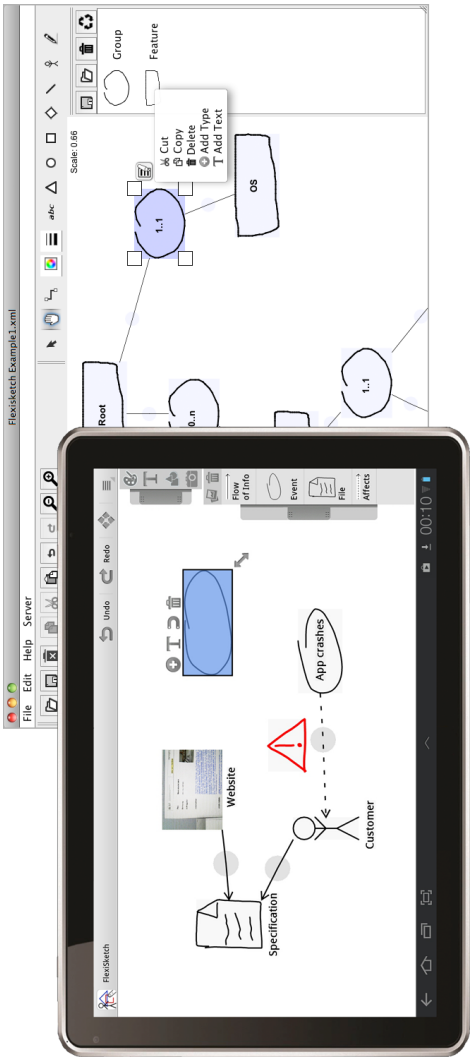


Figure 6.1: Screenshots of the mobile and desktop versions of FlexiSketch showing the UIs and some model sketches.

devices. After startup, the tool presents a white drawing canvas, inviting for free-form sketching. Upon lifting the finger from the screen for a certain amount of time, FlexiSketch converts the drawn lines into an individual symbol. When the user draws a line from one symbol to another, the tool converts the line into a link between the symbols. A symbol can be user-drawn, an imported shape, or an image. Additionally, the user can add text boxes to elements.

Symbols and links can be selected by tapping, whereupon they are highlighted in blue and reveal a context menu. A highlighted element can be dragged around on the canvas. The context menu allows users to manipulate elements in different ways, e.g., add text, scale, delete, and merge two symbols into one. The context menu of a link also allows a change of its appearance and a choice between a unidirectional and bidirectional link.

A folding menu on the right edge of the screen contains typical editor options like choosing stroke color and thickness. It also has options to add free-floating text boxes, existing images from the filesystem or camera, and provides six standard geometrical shapes including a square, a circle, and a stickman.

More details about the technical solution regarding sketching and a first usability study are presented in [WSG13a].

6.2.3 Lightweight Metamodeling and Export

FlexiSketch uses a *lightweight metamodeling* mechanism, supporting the definition of types for symbols and links, and the definition

of cardinalities for link types. Advanced metamodeling concepts such as inheritance, hierarchies or complex constraints are not supported. This is a deliberate decision, as FlexiSketch is not intended to support metamodeling experts in creating full-fledged modeling languages. Instead, our approach is intended for users with little or no metamodeling knowledge and should provide "just enough" metamodeling features that do not overwhelm those users. Also, users should not get a strong feeling that they are actually metamodeling; as much as possible should happen behind the scenes and should be transparent to the users. Our vision is to support the creation of a coarse metamodel which helps to add meaning to model sketches and also allows for their export/import, so that the sketched content can be refined with other, more formal modeling tools. Currently, the added meaning primarily exists in the type names (assuming that the users have a definition for the respective names in mind) and implicitly in the restrictions defined by the cardinality rules. In other words, we currently focus on storing the concrete and abstract syntax (the metamodel). The semantics is only stored implicitly in the form of names.

To assign types in FlexiSketch, the users first select an element (a symbol or link), and then tap on the *plus* icon in the context menu (see Figure 6.1). The appearing dialog then allows them to define and assign a type.

All type definitions are managed in a type library. This type library can be shown by unfolding a container on the right edge of the screen. The type library provides a drag and drop mechanism that allows users to create new instances of types on the drawing

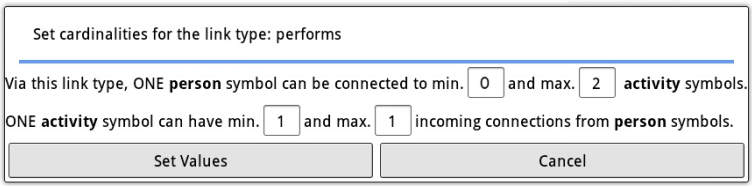


Figure 6.2: The cardinality dialog.

canvas (e.g., a user sketching a sequence of activities could draw and define an activity symbol once, and then get copies of the of the activity symbol via the drag and drop mechanism of the type library).

A *relationship type* is defined by a link type and the two connected symbol types. The user can define cardinalities of relationship types by selecting a link on the drawing canvas and using the corresponding context menu option to open the cardinality dialog (see Figure 6.2). There, minimum and maximum cardinalities can be defined for both directions. As a precondition for defining cardinalities of a relationship type, all elements of that relationship type must already have a type assigned (i.e., the link and the two connected symbols).

The main menu of FlexiSketch includes a wizard that checks whether all elements on the drawing canvas have types assigned. When the wizard is launched, it goes through all undefined elements step by step, highlighting and centering each element while asking for a type definition. The user can skip definitions or delete unwanted elements. In the last step, the wizard asks the user to define missing cardinalities.

A more comprehensive description of the metamodeling mechanisms of FlexiSketch can be found in [WSG13b].

Our tool exports two XML files, one contains the model sketch, the other contains the metamodel. If there is no metamodel information, the sketch XML file can still be used on its own, as it contains a structured list of the sketched elements (describing a generic graph consisting of nodes and edges). If there is a metamodel XML file, the entities from the sketch XML file link to it accordingly.

Furthermore, metamodels can be stored and exported independently from sketches. This allows the user to create and store simple modeling languages that can be re-used later on. Thus, there are two ways in which our tool can be used: i) the user starts to sketch without a metamodel, draws arbitrary node-and-edge diagrams, and adds metamodel information if/when needed, or ii) the user loads the metamodel of a previously defined modeling language and re-uses that language when creating new sketches – while still having the option to augment the language with new constructs on the fly.

The user has the option of converting the XML files to the GOPPRR format, which is used by the commercially available metamodeling tool MetaEdit+ [KLR96]. The sketched model and the metamodel can be imported in MetaEdit+. It is then possible to further refine and augment both artifacts. MetaEdit+ provides a palette with the symbol and link types that the user defined in FlexiSketch. MetaEdit+ also checks the defined cardinality

constraints when the user continues to work on the model. Figure 6.3 shows a model sketch in FlexiSketch and its corresponding exported version in MetaEdit+, including its metamodel.

6.2.4 Collaboration

Design Issues

For adding a collaboration mode to our tool, we identified five key design issues (D). These issues also reflect selected design guidelines that can be found in research about computer supported collaborative work [GG98, GG00, HLS+10, Tan91]:

D1: *All meeting participants should be able to edit the workspace simultaneously.* This fosters participation and can save time when work is performed in parallel [Tan91]. The guideline also implies that access to the workspace must be available at more than one single physical location, because restricted access (e.g., due to the limited physical space in front of the input device) can stifle participation [HLS+10].

D2: *A tool should prevent conflicting inputs from multiple participants.* This is a sub-issue of supporting the coordination between participants [GG00].

D3: *A tool should provide both shared and private views.* While a shared view helps to gather the foci of participants [GG98], private views allow users to create private notes [HLS+10].

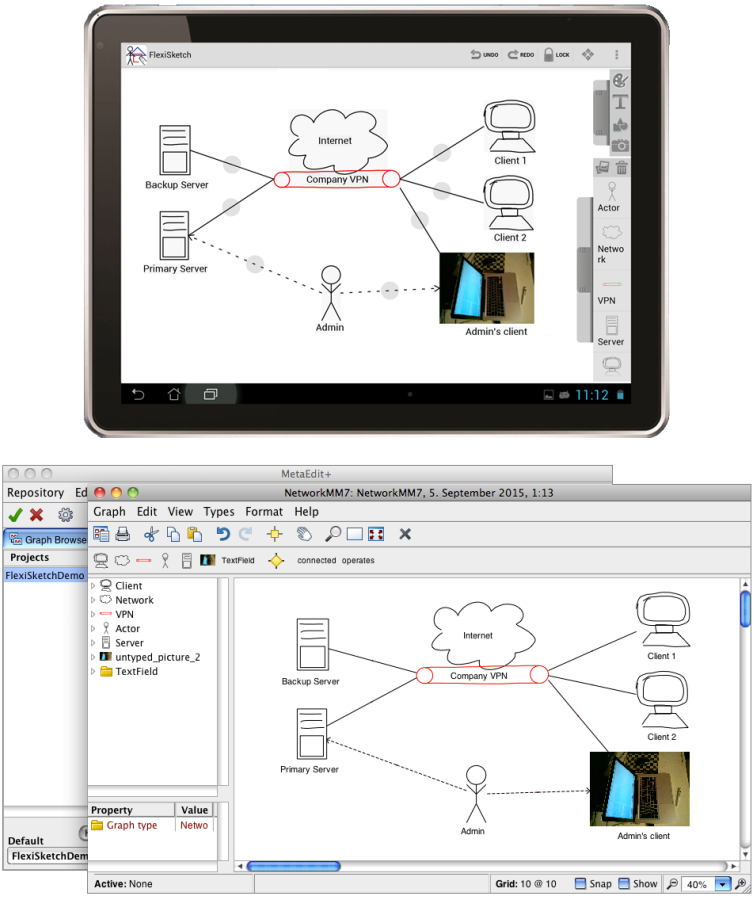


Figure 6.3: A model sketch and its metamodel exported to MetaEdit+.

D4: *All participants should immediately receive the results of a design session.* Since a meeting is an event embedded in a larger work context [HLS+10], it must be easy for all participants to take the meeting results with them for later re-use.

D5: *The tool should increase the awareness of each other's actions.* Participants should always be able to know what the other participants are currently doing [GG00]. The tool should actively support this in situations where the results of user actions cannot be seen within few seconds.

Due to time constraints, we prioritized the design issues and addressed D1 to D4, while D5 is left for future work (although it is partly addressed by the visualization of the locking mechanism – as described below). The resulting tool solution was sufficient for conducting our studies.

Implementation

FlexiSketch Desktop is a version of our tool suited for PCs and electronic whiteboards. The interface looks slightly different because it is adjusted for mouse and keyboard input (but everything is also accessible on touch screens by simple finger touches). However, its functionality is essentially identical to the mobile version.

Our tool supports multi-screen collaboration by using the desktop version as a server and connecting multiple tablets to it over Wi-Fi[2].

[2] A demo video is available at http://youtu.be/0kHjNfHLViM

Multiple persons can simultaneously work on the same sketch and together define a simple custom modeling language. All changes get synchronized immediately between the connected devices. While different parts of the sketch can be edited concurrently, a locking mechanism ensures that each part is only edited by one person at a time: as soon as a sketched element gets selected, this element becomes locked and appears with a red background for all other users. This mechanism prevents inconsistent states of individual elements (e.g., it cannot happen that one user deletes an element while another user is in the middle of assigning a type or adding text to the same element). Furthermore, highlighting locked parts of a sketch provides some user awareness, because a user can see the parts of a sketch that are currently being edited by other users. There is no specific indicator telling that another user is currently looking at a popup dialog for changing an element, and the change gets synchronized once the user closes the dialog. However, the element is locked and highlighted during that time, which indicates that a user is changing it in some way.

While the desktop version is not receiving any user input, it shows an overview of the whole sketch canvas and type definitions, and can optionally be projected onto a wall. Users can zoom and scroll their individual views on the tablets. A session moderator could steer discussions by using the desktop version to zoom in on certain parts of a sketch or to highlight some parts of it. Due to a technical limitation, the desktop version is currently read-only when it is in collaborative mode, and tablets connected over Wi-Fi must be used as input devices. We plan to change this in a future release of the tool.

A share function allows any user to push their workspace state to all other connected devices, or to pull the server state. This allows users to join a session at any time, receiving the current workspace state. Or, a user could disconnect at any time to have a private workspace, and re-connect again to share her work.

6.2.5 Target Audience and Field of Application

The target users of FlexiSketch are: (i) software or systems engineers who create sketches during a system development project, and (ii) requirements engineers (business analysts) who use sketches to create and communicate requirements. We expect that the modeling skills as well as metamodeling knowledge of our target audience vary significantly. We assume that FlexiSketch users may or may not have knowledge about or previous experience with metamodeling.

As FlexiSketch has been designed as an alternative to using paper and pencil or whiteboards, it aims at being applied in all situations where creativity and the generation and communication of ideas are central activities, with commensurate model sketches being created in the process. Typical settings are meetings, workshops and discussions with or among stakeholders where they create ideas, elicit requirements, and create early design solutions. We especially focus on early requirements engineering (RE) sessions. We believe that this is where informal sketches are most frequent [CVDK07, GD96]: early in the software process, and when external stakeholders such as customers are present (who might not know

standard modeling languages such as UML). As FlexiSketch runs on mobile devices, it can be used at any time and in any place where ideas come to one's mind. This also includes requirements elicitation in situ, i.e., when a stakeholder sketches ideas about requirements in the actual work environment.

6.3 Study 1: What patterns emerge when modelers collaboratively define modeling languages?

To our knowledge, FlexiSketch is the first software tool that enables the definition of modeling languages collaboratively in a sketching environment. Therefore, with RQ 1 we want to investigate how potential users of our approach collaborate during this activity. We performed a qualitative study consisting of simulated workshops with six small student and practitioner groups. Results of this study are useful for both i) improving our own approach, and ii) future work in the field of collaborative, tool-supported metamodeling. Related work about this topic is still scarce. The traditional view on metamodeling is that a single expert creates a new modeling language beforehand, whereupon modelers then start using it [Kle08]. In contrast, with our approach, modelers can create modeling languages on the fly while sketching models, which allows them to come up with languages that are suitable for their particular situations (i.e., for creative, early requirements eliciation meetings). They can change or augment existing languages on

demand. Because sketching and metamodeling are intertwined in our approach, we also need to analyze how groups sketch together in order to put their metamodeling behavior in the right context.

We chose both practitioners and students to have a mix between more and less experienced engineers. Eight Master students attending an advanced requirements engineering course at the University of Zurich participated in the study, as well as nine software and RE practitioners from different companies. Some of the students already worked in industry for several years. The simulated workshops were part of the course, but we told the students that this session's purpose is to evaluate our approach and does not affect their course grades. We divided the students into three groups of two or three people, SG1 (S1 and S2), SG2 (S3-S5), and SG3 (S6-S8). We believe this to be a realistic group size for ad-hoc meetings where participants come up with new ideas and create model sketches. Similarly, we had three practitioner groups, PG1 (P1-P3), PG2 (P4-P6), and PG3 (P7-P9). PG1 consisted of practitioners from different Swiss companies who knew each other from their time at the university. All of them have similar roles in their respective companies. PG2 consisted of members who work in an Austrian university but regularly receive tasks from industrial partners. P4 is not the direct boss of P5 and P6, but is one level above them in the hierarchy of the university. Members of PG3 work together within an Austrian company focused on mobile applications. P8 is the boss of P7 and P9.

6.3.1 Method

We could perform this study either by using only the desktop version of our tool on an electronic whiteboard, or the mobile version in a multi-screen setup. Since the mobile version stands for the core of our approach, and electronic whiteboards are not available everywhere, we decided to perform the study with the collaborative mobile tool and using PCs or projectors to display a shared overview of the workspace. Every workshop participant received an Android tablet with a screen size between 9.4 and 10.1 inches. The tablets were not identical, but we believe that the same is true in a real-world scenario where practitioners bring their own tablets. The practitioner workshops were conducted in German. Quotations from German speaking participants presented in this article were translated to English.

Every group sat around a table. For student groups, we placed a big screen showing the shared overview on every table. For practitioner groups, we were able to use existing projectors on-site. Figure 6.4 shows one of the practitioner groups during the study.

First, we gave a five minute introduction to FlexiSketch and every participant could try the tool in single-user mode for an additional five minutes. We then introduced the collaboration features and connected the tablets with the server. At that point, the main modeling task started (the task description[3] follows below). We

[3]The handouts and survey for the student groups are available at https://files.ifi.uzh.ch/rerg/flexisketch/StudentHandouts.pdf

Figure 6.4: A group of practitioners is working in one of our simulated workshops.

told all groups that they have to solve the modeling task in a collaborative way (but we did not say how; neither did we put restrictions on the seating, nor did we introduce a workshop moderator). A part of the task description stated that all elements of the sketch must have a type assigned at the end of the session. For twenty minutes (the time was controlled), the groups performed modeling and created type definitions for their modeling languages. Due to a technical limitation, cardinality rules were only inferred automatically. For the manual definition of cardinality rules, we refer to our second study in Section 6.4.

The practitioner groups were asked to think about and choose a current RE-related task or problem from their company[4], and they were free to choose or invent any modeling language. In order to

[4]The members of PG1 picked a task from one of their companies and turned it into a more general problem to which all group members could relate to.

have a similar initial situation for the student groups, we provided the students with a shared work context that otherwise would have been lacking compared to the practitioner groups: we predefined the task for the students, as well as the diagram types (but not the notations) they should use. The practitioner groups would most likely not think of completely new languages to model the selected RE tasks in our simulated workshops, but use something similar to what they have already used before in their daily work life (this is confirmed by our study results). We concluded that we can get a comparable effect in the student groups by telling them to use diagram types to which they got briefly introduced to in their previous studies, while (as novice modelers) being still far from becoming experts in understanding and using those diagram types. The students received two modeling tasks about a fictive online learning platform, each lasted for ten minutes. They had to draw a use case (UC) diagram, followed by a graphical user interface (GUI) for the use case "sign up on the online portal". The tasks were given in written form (in natural language) and also stated that students should be creative and add additional ideas if possible. Each group was supervised by one of the FlexiSketch creators who did not intervene unless there was a technical problem with the tool.

All sessions concluded with a semi-structured interview with the whole group. Because time during the course was limited, students were further asked to fill out an online survey. This also allowed each student to give individual and anonymous feedback. Seven students completed the survey.

Each group was video-recorded during the whole session. The experiment data we collected and analyzed includes video recordings of each group, FlexiSketch log files listing user actions with timestamps, and participants feedback from the semi-structured interviews and survey.

6.3.2 Analysis

Each video was analyzed in two iterations by the article's first author. In the first iteration, the editing behavior of each study participant was coded with a binary function (1 during the time when the participant is touching the tablet, 0 for the rest of the time). In order to filter out fine-scale structures while keeping the important behavioral patterns, we applied smoothing to the results by using discrete time-steps of two seconds. The same author coded the conversation between the participants in a second iteration. Starting with the experience from the first iteration, he created a coding scheme. Then, he refined the scheme by analyzing two of the video recordings. He discussed the coding scheme with the other authors and his research colleagues before finalizing it. The final scheme consists of four categories, and each utterance from the workshop participants was put in one of the categories: the *modeling* category contains utterances about the modeling task and the domain model (e.g., "*We have a further actor, professor, who can also upload documents*"). The *semantics* category includes statements and questions about types and the modeling language (e.g., "*What does this element mean?*", "*I'm*

going to draw and define an actor symbol"). The *tool* category contains utterances related to tool features and usability (e.g., "*Can symbols be rotated?*", "*You need to hold down the finger to drag and drop*"). The *other* category contains chatter unrelated to the task and tool, e.g., when someone mentions the weather or comments on the drawing skills of someone else.

The first author coded the utterances in all videos. Also, dual coding was performed on one of the videos by involving the second author as an additional coder, and its results were discussed. We synchronized the time-codes by allowing deviations of +/- two seconds. Then, we measured the inter-rater agreement, ignoring brief utterances that were coded by one author but ignored by the other one (utterances such as "*m-hm*", "*ok*", and "*oh*?"). Calculating Cohen's kappa resulted in a value of 0.79, which implies that the agreement level is between "substantial" and "almost perfect", and that the first author's coding possesses a reasonable validity.

After we finished the classification, we looked closer at the *semantics* category and investigated how these utterances relate to the type defining activities of the participants. We were interested in analyzing how participants communicated their type definitions (e.g., do they discuss or just notify each other about the types they create? Do they talk to their team members before or after creating a type?).

On some tablets, FlexiSketch did not generate log files due to a software bug. We used the log files from the remaining tablets to

triangulate the video data. We further analyzed the results from the semi-structured interviews and the survey, and we grouped statements to find accumulations and interesting patterns. We also looked for connections between participants' behavior during the experiment and their interview answers.

6.3.3 Results

Figure 6.5 shows extracts of the diagrams created by practitioners, together with the defined types[5]. Table 6.1 reveals how many elements and types are contained in the diagrams.

Table 6.1: The amount of symbols, links, and defined types contained in each diagram from students (left) and practitioners (right).

		#symb	#links	#types
SG1	UC	12	7	5
	GUI	13	0	4
SG2	UC	10	6	3
	GUI	3	0	1
SG3	UC	7	5	4
	GUI	8	0	5

	#symb	#links	#types
PG1	20	16	9
PG2	18	15	3
PG3	9	3	5

[5]The full diagrams can be found in high resolution at https://files.ifi.uzh.ch/rerg/flexisketch/TeamResults.pdf

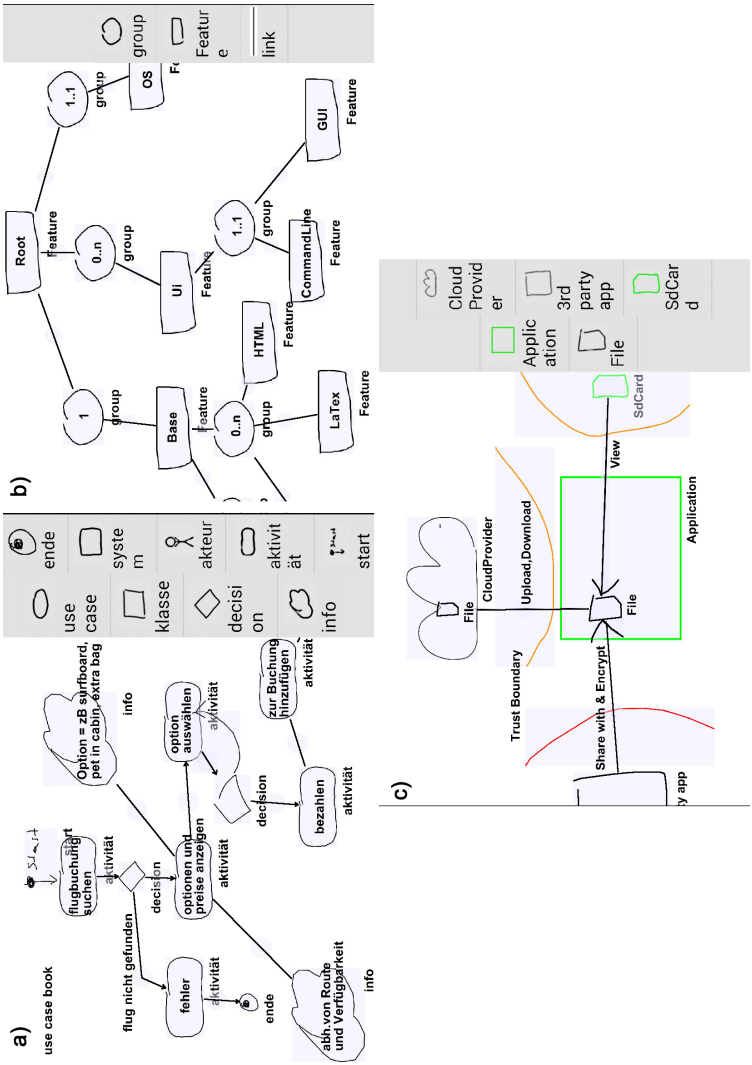


Figure 6.5: Extracts from the results of practitioner groups, a) PG1, b) PG2, c) PG3. The grey bars show the defined types.

R1.1: Phases of simultaneous editing happened in all groups. In all six groups, phases of silent, simultaneous editing and phases of discussions with or without editing were interleaving. This can be seen in Figures 6.6 and 6.7 which show when each group member was editing and/or talking. In the practitioner groups, all group members were simultaneously editing during 8.2% to 13.5% of the time, depending on the group. In the student groups, the values were higher with 20.1% and 23.3% for SG2 and SG3, and 51.5% for SG1 (the group of two). The different amount of simultaneous editing between practitioner and student groups correlates with the different amount of communication (see R1.2). We found two special cases regarding the editing behavior: practitioner P4 in PG2 made few edits, instead she contributed to the work by asking many explorative questions and proposed alternatives, e.g., "*Should we have different feature types or just one type called feature?*". Furthermore, we identified student S3 as a leader in SG2. He talked the most and came up with many modeling ideas, while the other members focused on sketching activities.

R1.2: Practitioners communicated more than students. We counted during how many of the discrete two-seconds time steps communication took place, and found that practitioner groups were talking for 12 minutes on average, while student groups were talking for 7.7 minutes on average. During the video analysis, we found that practitioners did communicate well: they frequently discussed about what to do, and informed each other about their current or next steps. Practitioners from all groups stated in the interview that they did not experience communication or

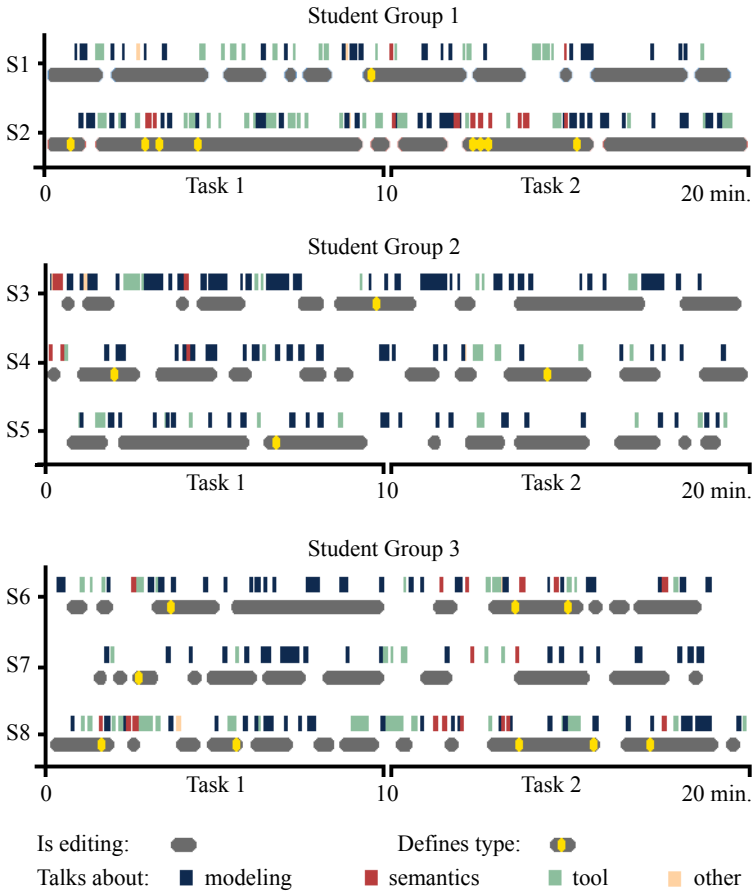


Figure 6.6: Phases of editing and discussions in student groups.

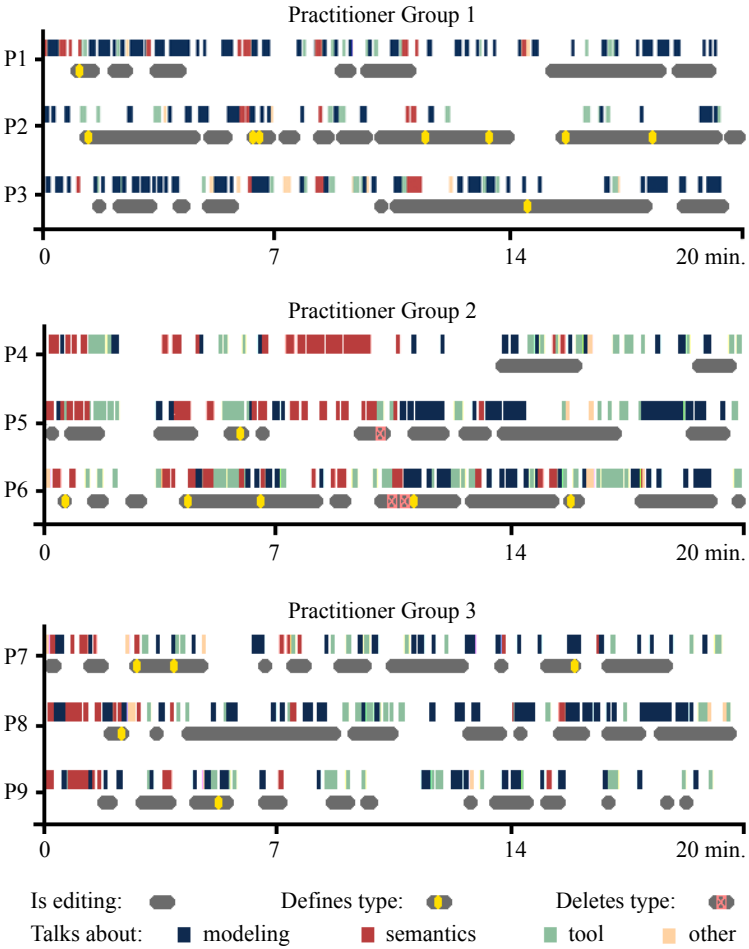


Figure 6.7: Phases of editing and discussions in practitioner groups.

coordination issues. No one disagreed. In contrast, students from SG1 and SG3 stated that they perceived a lack of coordination, because their attention was drawn to the interaction with the tool. They believed that this reduced the amount of discussions they had, e.g., S2 said: "*Especially at the beginning we did not talk, each of us was concentrating on his own tablet*", and S3: "*Each of us drew something. We only discussed after noticing that two of us had sketched the same thing and we needed to agree about what to keep and what to delete.*" Our video analysis confirmed these coordination issues. In this regard, P1 recognized an important difference between a pure sketching environment and our tool: "*When sketching collaboratively with a tool, you can just start to draw. But here, when defining types, you need to be more careful and coordinate.*" Sketching can always be done locally, while type definitions affect the whole workspace (they change the type library and are valid for the whole sketch canvas).

No student group started the task with a brainstorming or extended discussion. Instead, communication happened rather "incremental": multiple times during the session, students discussed what they are going to draw next, and who draws what part. These moments were followed by phases of silent editing.

In contrast to the students, the practitioners almost always informed the other group members about their next steps before drawing anything (e.g., P1: "*I'm going to draw a system boundary, okay?*"). One exception happened in PG1, where communication started to decrease during the session. Towards the end, the group members were simultaneously sketching three different types of

diagrams next to each other. To ensure consistency between diagrams, they discussed key elements which appeared in all diagrams, such as specific stakeholders and use cases.

All student groups tried to fit their diagrams on a single tablet screen (such that they always saw all parts without scrolling and zooming). S5 from SG2 stated: "*We wanted to make sure that we always see the changes made by each other, and that no change happens outside of a tablet's current view*". In contrast, diagrams from the practitioner groups clearly extended the size of a tablet screen (except the diagram from PG3 which needs a small amount of zooming to make it fit).

The different groups did use the big screen with the overview to significantly different degrees. While PG1 and PG3 barely looked at it (P1: "*We used it once or twice*"), the video analysis revealed that group members of PG2 used it many times to discuss the design and further steps with the help of a shared view. Feedback from PG2 confirmed this. Likewise, five of the seven students who filled out the survey were very positive or positive about the big screen with the overview (measured on a Likert scale).

Many participants peeked onto each other's tablet from time to time (the same behavior was found in a study from Loksa et al. [LMLvdH13]). This is true for all students as well as for P1, P2, and all practitioners from PG3. P1: "*It helps to coordinate, to see what the other person is doing*". Only practitioners from PG2 did not reveal this behavior. They were sitting a bit further apart from each other compared to the other groups, which made it hard

to see the screens of each other's tablets. Instead, the members of PG2 looked at the big screen more frequently than other groups to discuss the sketch.

R1.3: Notations were defined by multiple participants. In all simulated workshops, type definitions were created by multiple group members. Out of all participants, P4 was the only person who did not define any type. Students defined a total of 9 (SG1), 4 (SG2), and 9 (SG3) types, practitioners 9 (PG1), 3 (PG2), and 5 (PG3) types. The yellow dots in the editing bars in Figures 6.6 and 6.7 indicate type definitions.

The video analysis showed that there was no discussion related to the graphical representation of types in all student groups as well as PG1, with one exception in SG3: S6 communicated that he is about to assign the type *use case* to one of his drawn elements. S8 interrupted him and asked whether they should instead use a nice geometrical shape for the representation, whereupon S6 agreed. Apart from this, only practitioner groups PG2 and PG3 briefly discussed what the individual types should look like (see R1.6 for more discussion details).

R1.4: Notations were defined incrementally and continuously during the whole sessions. Most often, groups defined new types as soon as they introduced new elements in the diagram. In all groups, types were not only defined at the beginning, but the notation grew incrementally during the whole task (e.g., Figure 6.7 shows that group PG1 defined five out of the nine types during the second half of the session). Practitioner groups PG2 and PG3

discussed many semantics concerns in the early phase of the modeling task, and then continued with incremental discussions and ad-hoc notation definitions at various points in time. Other groups did not have semantics discussions at the start, but showed the same behavior of discussing the language incrementally.

R1.5: Participants based their notations on familiar concepts and symbols. All practitioner groups ended up with non-standard modeling languages which are more (PG1) or less (PG6) based on existing standards (such as UML or feature tree models). Participants from PG1 and PG2 stated in the interview that their groups chose and agreed on language concepts that were familiar to all group members, and then started to adapt and augment them as needed. For example, PG1 started with a basic UML activity diagram notation and then augmented it with an additional element type during the session, an *info* type that is represented with a cloud-like shape. As another example, PG2 started with the idea of a feature tree model. They did not augment it, but ended up with a simplified version of the model type. Finally, PG3 came up with a DSML. Thus, we experienced three cases: i) augmenting a standard notation, ii) simplifying it, and iii) coming up with a custom notation. The common result is that none of the practitioner groups did stick to an unaltered, standard notation.

R1.6: Discussions about semantics depended on the chosen language constructs. Basing custom languages on standards also simplified the task of achieving a mutual understanding about the meaning of individual elements. Figure 6.8 reveals that

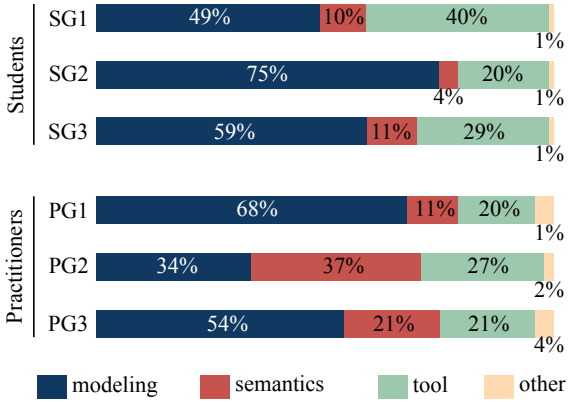


Figure 6.8: Talk category distribution in student and practitioner groups.

PG1, using a notation which was very similar to UML, devoted less of its discussions to semantics than PG2 and PG3. Four types in PG1 were defined by P2 without any discussion. PG2 discussed a lot about semantics and how they can map their concerns to model elements (they talked more about semantics than the concrete model). It was the only group that deleted some element types in the middle of the session and replaced them by a new one (see Figure 6.7). The recording of the discussion revealed that they found the old types to be at a wrong level of abstraction: at first, they did not define the type *group*, but multiple different subtypes of it. Later in the session, they decided that they do not need this level of detail, deleted the subtypes and replaced them by the single *group* type.

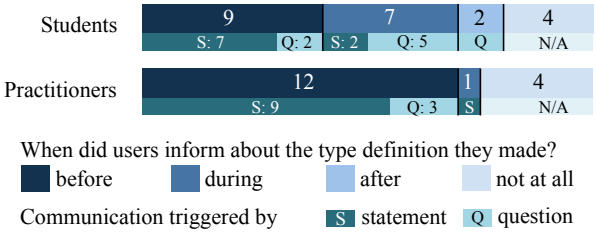


Figure 6.9: When participants talked about new types – before, during, or after defining them.

Student groups talked little about semantics. P2 mentioned: "*There was no need to discuss because we were all familiar with the use case diagram notation needed for the first task*". The resulting sketches show that all groups used the proper notation for use case diagrams. In contrast, the notation for the GUI in task two was not predetermined. While creating the GUI for task two, SG1 and SG3 talked more about semantics (see Figure 6.6), but SG2 almost completely neglected semantics. Indeed, they only defined one type during task two (Table 6.1).

Figure 6.9 reveals how participants communicated about type definitions. For nine out of 22 types, students discussed their type definition in advance (before they created it in the tool). In nine other cases, they only informed their group members by quickly mentioning the symbol type, either while inputting the types or only afterwards (for example, S6 and S8 mentioned at the moment when they hit the ok-button of the type definition dialog: "*Radio button*" and "*Text field*", respectively). Seven of these types were only discussed after another group member asked the creator of

a type what he/she is doing or what the newly appeared type means.

In contrast, practitioners discussed many type definition in advance (twelve types). Three of these discussion started with a question (e.g., P8: "*Should we define a type named file?*"), the others with a statement. No one felt the need to ask a question about an already defined type, except in PG2, where types got re-discussed after they were in use for some time.

Both student and practitioner groups did not discuss four type definitions at all. For practitioner groups, these four cases happened in PG1. After agreeing on the UML activity diagram to start with, they did not feel the need to discuss some types individually. P1: "*Borrowing most of the elements from UML allowed us to get a shared understanding of the symbols' meaning with little effort*".

R1.7: All groups created consistent notations. The survey revealed that the drag and drop mechanism for re-using types was liked a lot by the study participants. The video analysis and the resulting diagrams made clear that students and practitioners used the mechanism whenever possible (with one exception in PG1: Figure 6.5 shows two slightly different hand-drawn symbols for the type *decision*). Making heavy use of the drag and drop mechanism resulted in all participants using the same notation within a group, and therefore led to consistent notations. Also, the video analysis and log files provided no evidence that temporary inconsistencies happened during the workshops. In the end, almost all diagrams

from students and practitioners showed a 1:1 mapping between element graphics and types. Apart from two exceptions, there were no inconsistencies such as either having two different shapes with the same type assigned, or two identical shapes with different types assigned. One exception happend in PG1, where two symbols with different types (*system* and *class*) cannot be distinguished by the sketch recognizer, because they look the same. P2, responsible for creating the type *class*, told us in the interview: "*I wanted to sketch a better class symbol, but refrained from it because the session was about to end*". Other than that, all symbols can be distinguished by form and/or color. The second exception happened because of the tool's inability to rotate symbols. PG3 needed multiple *trust boundary* symbols with different orientations. P8 stated that "*the possibility to re-use defined types is a big motivation for defining them*". Not being able to rotate the symbol, P8 had no motivation for creating a type definition: "*Can I rotate a symbol? ... No? ... In that case, I do not need to assign a type to this particular symbol*". However, P5 and P9 stated another advantage of defining types: assigning types to symbols can serve as a kind of documentation and contribute towards the comprehensibility of a sketch. P5: "*Due to the type definitions, I think I will have less effort in understanding a sketch when I look at it again after several weeks or months*".

On a Likert scale, all students were very positive about the drag and drop mechanism. S2 said: "*As soon as you start to make bigger sketches, dragging elements [from the type library] onto the canvas is faster than drawing them by hand each time*".

Regarding completeness, PG3 did not define the *trust boundary* symbol, and SG2 did neglect definitions for the GUI elements. Apart from these cases, the type definitions for symbols were complete, i.e., every symbol in the sketches had a type assigned to it. In contrast, PG2 was the only group that defined a link type. Other groups neglected link types. We discuss possible explanations for this in the discussion Section 6.5.1.

6.4 Study 2: Can modelers define lightweight modeling languages correctly and completely?

With RQ 2, we want to investigate whether potential users of our approach manage to define modeling languages correctly and completely on their own. In a study, we conducted a quantitative experiment with 107 first-year undergraduate students from two universities (67 from the University of Zurich and 40 from the University of Applied Sciences and Arts Northwestern Switzerland FHNW) and complemented it with a qualitative experiment including eleven practitioners from different companies. The main goal of the student experiment was to find out whether students provide correct and complete lightweight metamodel definitions when using the metamodeling mechanisms of our approach. If they manage to do this, it would mean that we can create this kind of lightweight metamodels without the help of metamodeling experts. The main goal of the practitioner experiment was to see how useful

Table 6.2: Practitioner demographics: their work field (SE/RE), the years of work experience, experience with touch devices ((H)igh/(M)edium/(L)ow), and metamodeling knowledge (Yes/No).

Practitioner	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11
Work field	SE	RE	RE	RE	SE	RE	RE	SE	SE	SE	RE
Years of exp.	10	15	>30	8	2	2	11	23	9	4	3
Touch exp.	H	M	L	M	H	M	M	H	M	L	M
MM exp.	Yes	Yes	Yes	No	No	No	Yes	No	Yes	No	No

practitioners find our tool, and how well they manage to apply our approach. The practitioners have various degrees of modeling and metamodeling experience. Table 6.2 provides an overview of the practitioner demographics. Five practitioners work as software engineers and mainly conduct software design, architecture and development. The other six are specialized in requirements engineering. The practitioners had a broad range of experiences in SE and RE, including practitioners who were working in software and consultant companies for about two years after finishing their studies, as well as a practitioner aged 60, having more than 30 years of experience. Five practitioners had experience with metamodeling, and four of them had already used metamodeling explicitly in their jobs. The other six had no metamodeling knowledge. The students had basic modeling and no metamodeling knowledge, thus can be seen as novice modelers. They can be considered to be future software and requirements engineering practitioners, and therefore our future target audience. Furthermore, their level of modeling knowledge might be comparable to those of many business stakeholders.

6.4.1 Method

To be able to assess the quality of the students' solutions in terms of correctness and completeness, we specified the modeling task and the language that students had to use. This enabled us to compare their solutions to a ground truth (which represents an expert solution).

In contrast, in order not to limit our experiments to specific modeling languages and tasks, practitioners were free to choose their own languages and could model aspects of projects they were currently working on.

Student Experiment

Setup. From the 107 students, 64 students solved the experiment tasks with the mobile version of our tool on an Android tablet, while we randomly assigned 43 of them to a control group where they solved the same tasks with paper and pencil. Since we do not only want to evaluate our tool but also our approach, this would allow us to identify a possible bias introduced by the tool itself.

The experiment lasted 60 minutes and was structured into three parts: briefing, evaluation and debriefing. The briefing started with a short oral introduction to inform the students about their modeling task. In addition, the students got handouts which

provided a textual description of the example they were asked to model (the material is available online[6]).

For the students who used FlexiSketch, the handouts also included a short hands-on tutorial to get to know the tool's functionality. No introduction to metamodeling was included. The handouts only stated that the tool must be able to interpret the sketched diagrams, and that therefore all model constructs must be defined. It was made explicit that there is a passive wizard that can help to define all model elements and cardinalities.

The students who did the experiment on paper had to solve the same examples without tool support. The handouts stated that they should define all modeling constructs so that a person without knowledge about these diagram types can understand them. Their task was divided into three parts. Each part was presented on a new page: first, students had to sketch the diagram on paper. Second, they had to build a table with all sketched symbols and links in one column, and proper type names in another column. Finally, they had to document cardinalities for each defined link type.

During the actual evaluation, students modeled UML diagrams based on the given textual scenarios and provided metamodel information by defining symbol types, link types, and cardinalities. The students from the University of Applied Sciences and Arts Northwestern Switzerland FHNW had to model a use case diagram about a simple railway scenario including two actors, four use cases,

[6] https://files.ifi.uzh.ch/rerg/flexisketch/Handouts.pdf

Table 6.3: Amount of data points received.

	With FlexiSketch	With paper
Class diagram	34	33
Use case diagram	30	10

and three types of relationships: association, includes, and extends. The students from the University of Zurich had to model a class diagram based on a related railway scenario. This included five classes, two note objects as introduced in the handouts, and three types of relationships: association, inheritance, and describes. The latter was used to connect a note object with a class.

In total we received 107 diagrams including metamodel information, as summarized in Table 6.3.

At the end, all students were asked to fill out an online questionnaire for debriefing purposes[7]. The questionnaire included questions on their personal skills and their experience during the evaluation task.

Data Analysis. In a first step we assessed the quality of a student's model, comparing it to our ground truth[8] in terms of completeness and correctness (we created the ground truth for the models and the metamodels based on our textual scenarios). Since our lightweight metamodeling approach consists of metamodeling by example, a wrong model can affect the resulting metamodel. We wanted to take these aftereffects into account, in order to

[7] https://files.ifi.uzh.ch/rerg/flexisketch/Questionnaire.pdf
[8] https://files.ifi.uzh.ch/rerg/flexisketch/GroundTruth.pdf

receive values for the metamodel quality that are independent of the model quality. We differentiated between syntax and semantics. A semantically incomplete or incorrect model can still lead to the same metamodel if the syntax is complete and correct. But if there is a problem with the syntax, the metamodel will look differently. In cases where the student's syntax differed from our ground truth, we evaluated the student's metamodel against his/her model (and not the model from the ground truth) to rule out consequential errors. With this approach, we also respect that there might be other correct model solutions for the given task, apart from our ground truth.

We then measured the completeness and correctness of a student's metamodel by comparing it to our ground truth. We independently looked at type definitions and cardinality definitions for measuring completeness. To measure type definition completeness, we counted how many of the symbol types and link types found in a student solution were defined by that student (#DefinedTypes/#TypesInSketch). For the completeness of cardinality definitions, we checked how many of the link types that were defined by a student also had cardinalities assigned. Due to how our tool works, each link type has either none or four cardinalities assigned (minimum and maximum values for both ends of the link; if the user does not define all four cardinalities, the tool infers the remaining ones). We then counted how many of the student's definitions were correct. For symbol and link types, we considered them to be correctly defined if the wording either matches the official UML definitions, or if synonyms were used that have the same meaning (as judged by two of the article's authors). For car-

dinalities to be correct, the values had to exactly match our ground truth. For the correctness metric, we ignored type definitions for additional model elements that were introduced by a student if neither the sketched model element nor the accompanying type definition are part of our ground truth for the model and the metamodel (15 out of 107 students each introduced one such model element and respective type). For example, one student defined the additional type *extension point* for the use case diagram, and used one instance of it in the model. Assessing the correctness of such additionally introduced types would be difficult in many cases (e.g., when the types do not match those of the official UML standard), because the intention of the modeling person would have to be taken into account, which can lead to highly subjective judgements.

Apart from the models and metamodels which were part of the student solutions, we also investigated the log files that FlexiSketch produced during the experiment. The logs show a list of user actions with timestamps. This allowed us to see, e.g., if the app crashed, how many times a student used a functionality, and if the same symbol was re-defined several times. This kind of information cannot be deduced from the final student solutions. In combination with the student answers to the online questionnaire, this allowed us to reason to a certain extent whether some errors in the student's solutions are due to a lack of the student's knowledge or usability problems of FlexiSketch.

Practitioner Experiment

Setup. The setup was similar to the student experiment in order to allow for a comparison of the results. However, practitioners were asked to sketch a model about a real-world task or problem they currently experience in one of their companies' projects, and to use any node-and-edge notations – as they see fit – for the model. This could include established notations such as UML, but we emphasized the tool's openness towards custom notations in the introduction.

We first introduced FlexiSketch in a semi-structured way, following the tutorial from the student experiment. We then asked the practitioners to sketch a model that is typical for their model sketching activities in the context of their job. They should furthermore define all types and cardinalities.

Because very few students used our tool's wizard in the student experiment (see Section 6.4.2), we explicitly advised practitioners to consult the wizard after finishing their modeling activity. After the modeling task, we conducted a semi-structured interview where we asked the questions from the student experiment questionnaire. Additionally, we asked more specific questions about the usability and utility of the tool in the context of the practitioners' work.

Data Analysis. As the practitioners used their own custom modeling notations, creating the ground truth for assessing the correctness of the created metamodels is not possible, or at least

very difficult. But this was not the main goal of the practitioner experiment. Instead, we were interested to see whether practitioners think that our approach is useful for the types of modeling notations that they use (having a reality check for our tool), and whether they face similar difficulties compared to students, or if they have no problems in providing metamodel information.

Nevertheless, we also tried to achieve a best approximation to a metamodel ground truth (i.e., the practitioners' true intentions about the meaning of what they had sketched) by discussing the experiment results (e.g., the derived metamodels) with the practitioners. This delivered some pointers for judging metamodel correctness, but inherently contains partly subjective views (which might also change over time). We also asked the practitioners to draw their intended metamodel on paper. In contrast, we counted the amount of defined and undefined symbols to have an objective measure for metamodel completeness.

Most of the presented results from this experiment are based on an analysis of the semi-structured interviews.

6.4.2 Results

Students – Modeling Novices

R2.1: Models were of good quality, although not all were complete. In general, most students delivered models of good

Table 6.4: Completeness and correctness of student models and metamodels (without cardinality rules).

Diagram type		Model %		Metamodel %	
		Compl.	Corr.	Compl.	Corr.
Class	Paper	82.1	93.9	77.4	85.8
	FlexiSketch	66.1	95.8	73.8	60.4
Use case	Paper	77.8	89.0	95.6	83.5
	FlexiSketch	77.4	85.3	70.6	95.3

Table 6.5: Statistical significance of the differences between paper and FlexiSketch results.

p-values	Model		Metamodel	
	Compl.	Corr.	Compl.	Corr.
Class diagram	0.002	0.38	0.64	0.004
Use case diagram	0.94	0.50	0.00006	0.096

quality (as shown in Table 6.4). While many models were missing some parts (the average model completeness was 75.9 percent), most of the drawn elements were correct with an average of 91 percent.

Model results between paper and FlexiSketch exercises were similar, except for the completeness of the class diagrams (82.1% for paper solutions and 66.1% for FlexiSketch solutions). We performed a two-sided Welch's t-test and found that the difference is statistically significant (p-value: 0.002) for the class diagram completeness, while the other differences in model quality are not statistically significant (Table 6.5). On tablets, fewer students used the additionally introduced *note* symbol and the accompanying link type *note-link* when creating the class diagram (this can be

seen in Figure 6.10). Also, some students with tablets depicted association and inheritance relationships with the same visual link style.

For the use case diagram, many students did not sketch all association relations between actors and use cases, which reduced model completeness (for both tablet and paper solutions).

R2.2: Metamodel quality differed between type and cardinality definitions. Table 6.4 shows the average completeness and correctness of the metamodels. These values were calculated by counting symbol and link types, but not cardinalities. Students' metamodeling performance varied significantly between element types and cardinality definitions. Therefore we look at these results independently (type definition results are covered in R2.3 to R2.5, cardinality definitions in R2.6).

R2.3: Metamodels were more complete on paper. Table 6.4 reveals that metamodel definitions done on paper tended to be more complete. The difference for the class diagrams is not statistically significant (p: 0.64), while there is a significant difference for the use case diagrams (p: 0.00006, Table 6.5). Many students using FlexiSketch did omit a definition for the *association* relationship in the use case diagram, which is the main cause for this difference. Furthermore, the *actor* type is lacking in some solutions.

In terms of completeness of the FlexiSketch solutions, we were not only interested as to whether all types were defined and appeared in

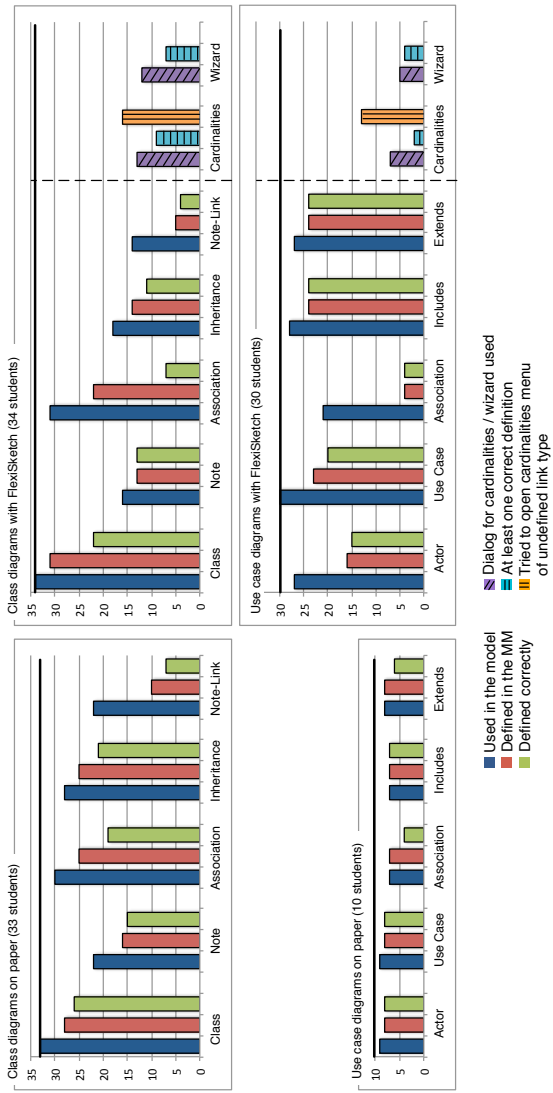


Figure 6.10: Amount of metamodel information defined by students.

the type library, but also whether all elements on the sketch canvas had a type assigned. FlexiSketch log files show that most students defined an element type right after sketching a new element for the first time. Afterwards, they used the drag and drop mechanism of our tool to create more instances of the same type (instances created with drag and drop automatically have a type assigned to them). Therefore, in most of the FlexiSketch solutions, all elements on the sketch canvas were defined. Students who did not use the drag and drop functionality assigned the type to each instance by hand. There was only one solution where some instances of the same type did not have the type assigned to them (i.e., an incomplete assignment of one particular type to all its instances). For the rest, either all instances of a type were correctly typed, or none of them had a type assigned in the first place.

R2.4: Metamodel correctness varied between diagram types. Evaluating the correctness of the different metamodels resulted in diverse values.

Figure 6.10 presents a more detailed view of the completeness and correctness of students' metamodel definitions. For every element type, it shows from left to right: how many students sketched this type in the diagram (blue), how many students provided a definition for the element type (red), and how many of these definitions were correct (green). Complete and correct type definitions would mean that all three bars of a group have the same height. For the students who used FlexiSketch, the figure also shows how many students used the cardinality dialog and the wizard at least once (violet), and how many provided at least one

correct definition with the respective dialogs (cyan). Additionally, the amount of students who tried to define cardinalities for an untyped link is shown (orange).

While students with FlexiSketch performed very well in defining use case diagram elements (95.3% correctness on average; the second and third bar of each group in Figure 6.10 have similar or same heights), students defining class diagrams with FlexiSketch were less successful (60.4% correctness). 15 out of the 22 students who did provide a definition for the *association* relationship did provide a wrong definition, resulting in the biggest difference between red and green bars in Figure 6.10. For example, some of these students named the relationship simply *arrow*, or they defined it on the model level as *serves*, which only makes sense for that particular instance of the association relationship. This problem was less present for students who did it on paper (six out of 25 students). There is a clear significance regarding the difference in paper and FlexiSketch metamodeling results for class diagrams (p: 0.004). On the other side, FlexiSketch results for use case diagrams are better than paper results, although not statistically significant (p: 0.096) unless we lower the level of confidence to 90 percent.

We also counted how many of the definitions correctly describe element types on the metamodel level (independent of whether it is the correct type name or not), and how many of the definitions were instead done on the model level, describing particular instances of the types (e.g., having types named *serves* and *contains*, instead of having a type definition named *association* for the respective relationship type in the class diagram). Regarding the use case diagram, 33 out of 38 definitions on paper and 89 out of 91 definitions

created with FlexiSketch were on the metamodel level. For the class diagram, 99 out of 104 definitions on paper and 68 out of 85 definitions created with FlexiSketch were on the metamodel level. Many of the remaining 17 definitions were describing instances of association relationships rather than the relationship itself. Six students reported in the post-experiment survey that they had problems with assigning different types and/or cardinalities to different instances of the same relationship type. This confirms that they did not think on the metamodel level in these cases.

R2.5: Students with tablets provided more accurate type names. As far as the accurate naming of types is concerned, FlexiSketch solutions tended to be superior (p: 0.037) to paper solutions: on paper, more students did not provide the exact type names but variations and paraphrases (e.g., on tablets, the stickman type was almost always named "actor", while on paper it was sometimes named as "user" or "system user". Also, instead of using the precise names for associations, they were sometimes paraphrased, e.g., "arrow for extends relationship" instead of "extends"). Taking into account only correct type definitions, tool solutions yielded 125 precise definitions out of 144 (86.8%), while paper solutions yielded 86 precise definitions out of 124 (69.4%).

R2.6: The cardinality and wizard dialogs were rarely used. Figures 6.10 and 6.11 show that the cardinality dialog and the tool wizard were used by relatively few students. 13 students modeling the use case diagram and 16 students modeling the class diagram tried to access the cardinality dialog before they defined the respective link type. This is something that the tool

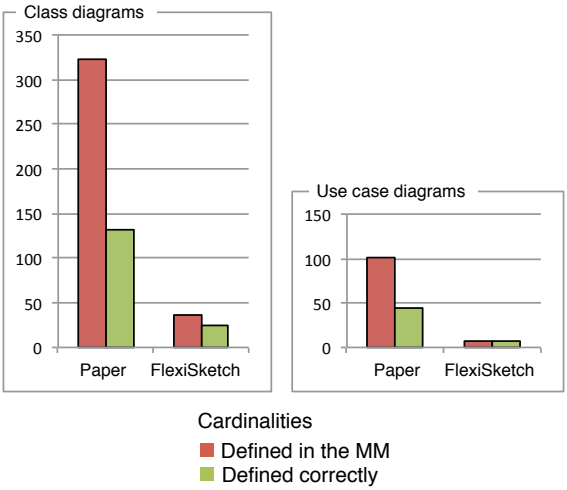


Figure 6.11: Amount of cardinalities defined by students.

does not allow – the link type needs to be defined before cardinalities can be specified. This means that the number of students who used the cardinality dialog would have been significantly higher if this tool limitation would not have existed. Whether these students would have provided correct cardinalities or not remains unknown.

Most students who did the experiment on paper provided cardinalities. For the use case diagram, 44.1% of the cardinalities were specified correctly. Three out of ten students provided cardinalities for link instances instead of link types. For the class diagram, 40.7% of the provided cardinalities were correct. 12 out of 33 students provided cardinalities for link instances. With FlexiSketch, much fewer students (about eight times less) defined

cardinalities. Therefore, the red bars in Figure 6.11 are much lower for FlexiSketch solutions compared to paper solutions. However, 75% of the cardinalities provided with FlexiSketch were specified correctly, which is a much better correctness value compared to paper. But this result could be biased by the inability of many students to access the cardinality dialog (i.e., if we assume that only the best students managed to access the cardinality dialog, then it is not surprising that we obtained a better correctness value).

In total, only 17 out of 64 students consulted the wizard during the experiment. Some students reported in the post-experiment survey that the wizard is not needed, because everything can be done without it. Others said that they simply forgot about the wizard's existence.

R2.7: No time difference between tool and paper emerged. Before and after their main task, students had to write down the current time. Analyzing the times, we found no significant differences between the averages of paper and FlexiSketch solutions, with p-values of 0.52 (class diagram) and 0.86 (use case diagram), as shown in Table 6.6.

Practitioners – Expert Modelers

The practitioners drew various models, resembling use case, context, component and activity diagrams, but overall being simpler

Table 6.6: Comparison of times between students working with paper and FlexiSketch.

Times		Average	StdDev	p-value
Class diagram	Paper	0:22:11	0:08:30	0.52
	FlexiSketch	0:20:43	0:07:37	
Use case diag.	Paper	0:20:06	0:06:18	0.86
	FlexiSketch	0:19:39	0:07:32	

in their notation compared to UML standards. Three sketches combined process diagrams with static views such as use case diagrams. One model only consisted of boxes and arrows (as judged by its creator). We were expecting this kind of simple notations. This result is inline with a known design behavior, stating that "*designers draw what they need, and no more*" [MLPvdH14]. Moreover, a notation that is too complex can stifle creativity [CVDK07].

R2.8: Practitioners outperformed students considerably in terms of metamodel completeness and correctness. The practitioners have more (meta-)modeling experience than the students and showed a faster understanding of the respective tool features. Metamodels from practitioners turned out to be superior compared to students' metamodels in both completeness and correctness. Regarding correctness, we only identified mistakes in three metamodels: one practitioner failed to define element types on the metamodel level. Another practitioner used the same link appearance to depict three different relations, therefore the type definitions were not correct. Furthermore, one practitioner did not care about the semantics of a link and simply assigned the type *arrow*.

Table 6.7: Completeness (%) of (p)ractitioners' metamodels before (v1) and after (v2) consulting the wizard.

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11
v1	80	100	66.7	75	100	25	33.3	60	0	57	75
v2	100	100	100	100	100	100	80	100	100	100	100

Five practitioners handled type definitions the same way as the students in our previous experiment: they defined elements when they sketched them for the first time. Afterwards, they used the drag and drop mechanism to create more elements of the same type. Therefore, all elements of the same type were already automatically defined.

We told all practitioners to use the wizard at the end of the modeling task. But before we said this, we stored the solutions from the practitioners, such that we could compare these results with the student results (and not the results that include the improvements with the help from the wizard). The completeness of metamodels before practitioners used the wizard included the full range from 0% to 100%, as shown in Table 6.7. After consulting the wizard, only one metamodel was not complete. All symbols in the sketches were defined, except in one case, where a practitioner drew about 20 symbols conveying the same meaning by hand, and then did not add types to all of them.

R2.9 Different opinions about the usefulness of metamodel features. Almost all practitioners tried to define cardinalities. Seven out of eleven practitioners succeeded in providing 100% correct cardinalities. Four practitioners stated that the ability to

define cardinalities is relevant, while the others stated they do not need cardinality definitions for how they would use our tool in practice. They said that it is more important to assign meanings to the sketches in the form of element types in order to discuss the sketches with coworkers and business stakeholders and to reach a common understanding, rather than providing all necessary data for completely formalizing the sketches.

Six out of eleven practitioners said that the wizard is useful and gave tips for improvements, while five did not feel the need for a wizard. On the other hand, all but two did in fact use the wizard to define at least one additional element type. One practitioner preferred to use the wizard for all metamodel definitions, and therefore did not use the context menu icons of sketched elements.

Practitioners said that the wizard can be useful to complete the metamodel, to delete superfluous elements from the sketching canvas, and to scrutinize the sketched model.

6.5 Discussion

In this section, we discuss the results and answers to the two research questions, summarize the identified strengths and weaknesses of FlexiSketch, and mention threats to validity.

6.5.1 RQ 1: What patterns emerge when modelers collaboratively define lightweight modeling languages?

Results from the first study (Section 6.3) show that all participants except one took an active part in defining element types (R1.3). Participants oriented themselves by standard modeling languages but introduced additional types as needed (R1.5). Dekel and Herbsleb found the same, deliberate deviation from standard languages in their study [DH07]. The modeling languages were further created incrementally during the whole workshop sessions, and discussions about semantics happened during the whole sessions (R1.4). This behavior is exactly what our approach is meant to support and foster. Furthermore, we experienced that phases of simultaneous, silent editing did interleave with phases of discussion (R1.1 and R1.2). During the phases of silent editing, every participant was working on another part of the sketch. Then, during phases of discussion, participants explained to each other what they did, or were discussing next steps. Other studies have found the same design behavior of group members switching between synchronous and asynchronous work [MLPvdH14].

The results also show that user awareness is an important matter in a multi-screen setup even for same-place collaboration (R1.2). This is something we neglected thus far, except for the visualization of the locking mechanism. Indeed, students sometimes experienced communication and coordination issues because they were mainly concentrating on the tool. In other words, students did not always succeed in managing both the cognitive and the social space at the same time [LTH12]. Shih et al. [SNH+09] confirm our results and state that users do not automatically "develop a sense of tolerance for lack of social awareness" in collocated sessions. However, studies suggest that people can learn how to cope with a multi-space setting [LMLvdH13]. One effect of this is that students usually discussed element types only after they defined them (because one of the students asked for clarification), while practitioner groups talked about many type definitions in advance (R1.2). Indeed, practitioners did not report coordination problems, and our video analysis shows that they were able to focus on both the communication and their individual screens. However, they mentioned that splitting the focus does require additional cognitive effort.

Apart from user awareness features, a separate overview – similar to the one we used in the study – can reduce coordination issues[GG98]. However, in order to reduce the amount of focus points (the personal screen, the overview, and the communication), it would make sense to incorporate the overview on the same screen as the personal view. With the small screen sizes of tablets, this in turn leads to a tradeoff regarding screen space and asks for new solutions [GG98]. Therefore, a single big screen can be preferable

if the infrastructure allows it, e.g., FlexiSketch Desktop can be used if an electronic whiteboard is available. If mobility becomes more important, users can switch to the collaborative version of the mobile tool.

The frequent use of the drag and drop mechanism had some positive effects on the results. First, participants were motivated to define a type as soon as they drew a first instance of that type, and then used the drag and drop mechanism whenever possible, which led to diagrams containing consistent notations (a 1:1 mapping between drawn symbols and symbol types, R1.7). We also assume that a frequent use of the drag and drop mechanism fosters diagrams that are defined more completely, since each re-used element already has a type assigned, compared to the scenario where a user draws a lot by hand (however, a future study has to confirm or reject this hypothesis). On the other side, the drag and drop mechanism also had a side-effect: participants committed to element types early. Related studies [DH07, OJDB10] show that the meanings of symbols are re-discussed and changed during design sketching on physical media. This did not happen often in our study, neither in the student groups with UML, nor in the practitioner groups where participants could freely choose their modeling language. Maybe the drag and drop mechanism tempted the participants to a premature commitment regarding the meanings of drawn elements [OJDB10] (however, the relatively short duration of our simulated workshops could be another explanation why participants did not re-discuss element types). Therefore, features such as the typing mechanism can have both positive and negative effects: they can

foster discussions about types and thus creativity, but they can also distract from the sketching task and stifle creativity.

In contrast to defining symbols, only one group (PG2) defined a link type. Possible reasons could be that link types cannot be dragged and dropped, and that FlexiSketch implicitly keeps a 1:1 mapping by regarding all links with the same appearance as being of the same (undefined) type (unlike symbols, users cannot define the appearances of links with free-form drawings, but they select link appearances from a given list). Overall, our tool is an example of how a sketching tool can help to have consistent and unambiguous sketches at the end of a session if the users want this.

Compared to the quantitative study where students worked individually (Section 6.4), type definitions from groups were much more complete. Whether this is because participants in the simulated workshops were more aware of their task, or because they were working in groups, remains open and cannot be answered with the data we have.

6.5.2 RQ 2: Can modelers define lightweight modeling languages correctly and completely?

Results of our second study (Section 6.4) show that practitioners had no problems using our metamodeling mechanisms. In contrast, students were able to define types correctly, but sometimes failed to distinguish between the model and the metamodel level when adding cardinality rules (R2.2).

Students – Novice Modelers

Model Quality. In general, the correctness of the drawn models was very high (R2.1). This shows that the students knew the notations and how to model. The lower values for use case diagrams are mainly an effect of students making the arrows of *includes* and *extends* relationships on the wrong end of the links. Analyzing the resulting sketches, we can conclude that most of the omissions and errors done in the models have little effect on the ground truth for students' metamodels.

Metamodel Quality. Regarding the metamodel quality, we have to distinguish between element types and cardinality rules. The students performed relatively well in defining element types. The difference in completeness between paper and the tool (R2.3) could be explained by the handouts for the students. The paper version presented each task on a new page (modeling, type definitions, and cardinality definitions) with empty space for solutions, which might have caused these students to care more about providing complete definitions. In contrast, the tablet version of the handout did not have this detailed structure, but mentioned that the wizard can be used to ensure completeness of the language definition. The wizard would then sequentially go through all missing definitions, providing the students a similar help compared to the paper handouts which listed the tasks on separate pages. However, many students did not use the wizard (R2.6). Possible reasons could be that they did not care much, or that they were too self-confident and did not think that they would need the wizard. By not using the wizard, students with tablets also missed the opportunity to

be reminded of cardinality rules definitions. Therefore, we suspect that our handout text was flawed by not urging the use of the wizard enough, which led to most of the differences between paper and tool solutions.

Figure 6.10 shows that students with our tool in particular had problems with defining the *association* relationships in both diagram types. A possible reason for the few defined *association* relationships in use case diagrams is that textbook examples depict *association* links between actors and use cases as simple lines without text, while *includes* and *extends* relationships are usually annotated with the respective names. Thus, students might not have been aware of the correct type name, or that they should assign a type at all to the *association* relationship. A similar argument could explain the difference between the amount of correctly defined *actor* and *use case* symbols: the name "use case" is more present as it is also the name of the diagram. The class diagram contained an additional pitfall compared to the use case diagrams, which caused a bit of confusion among the students with tablets (R2.4): they had to use the same link type (the *association* relationship) multiple times in the diagram, and add a different text label to each instance. Many students tried to add the text by defining the link type, rather than using the separate text box functionality. Therefore, students modeling the class diagram with FlexiSketch had more definitions on the model level compared to students who did it with paper. This could be improved in the future by improving the tool's usability and making the difference between the text box and the typing feature more clear.

Although some of the results were worse when compared to paper, we found two advantages of our tool over paper. First, the tool seemed to help keeping definitions short and precise (R2.5), so that more of them could be mapped 1:1 to the official UML names. In the future, this would facilitate the automation of an export/conversion of the sketch to other tools and languages. Maybe the students with tablets were aware of the fact that they are "teaching" the tool a language (as it was stated in the handouts), and therefore were motivated to give precise names. The precise wordings are also good for standardization and for reaching consensus in combination with an additional glossary that defines the terms. Second, the drag and drop functionality motivated students to define types and helped them to provide definitions on the metamodel level (although at the caveat of stifling creativity). When re-using types from the type library via drag and drop, it became clear to the students that re-using an element of type *class* is much more helpful for creating a class diagram than re-using an element of type, e.g., *train* or *train station*. Thus, the type library enabled the students to verify that their type definitions are at the right level of abstraction.

Such a verification mechanism does not exist when defining cardinality rules. Defining such rules had no visible effect on the modeling task, as the tool did not enforce the adherence to cardinality rules (however, our tool is designed with the idea that users can optionally enable a highlighting system that shows any violation of the cardinality rules). Moreover, many students were not reminded of defining cardinality rules because they did not use the wizard. In addition, the tool did not allow students to assign

cardinalities before assigning types to the link and the connected symbols (since cardinalities hold for specific relationship types)[9]. However, many students tried to open the cardinality dialog too early, and gave up in the process. These two problems might explain the difference in the amount of defined cardinalities with paper and our tool.

Overall, even the students with paper did not manage to provide correct cardinalities, having a success rate below 50%. We think that a software tool such as FlexiSketch could help the students in providing more correct definitions if it includes additional guidance, e.g., using *specification by example* as proposed by Qattous et al. [QGW10]. In a similar way, the guidance provided by the wizard could be improved.

Practitioners – Expert Modelers

From the solutions we can infer that practitioners' experience in thinking at different levels of abstraction contributed to the correctness of their metamodels (R2.8). Furthermore, it was interesting to see that almost all of the metamodels were more complete after practitioners had used the wizard, while half of the practitioners said that the wizard does not contribute substantially to their work (R2.9). This could mean that they did not feel the need for assigning types. Unfortunately, our study cannot prove whether the sketches would have become more concrete over time, and if

[9]This could be changed in future tool versions by using placeholders for the relationship types.

practitioners would then have wanted to assign types or define cardinalities. However, this is the beauty of FlexiSketch: it provides formalization capabilities if needed, but does not force the user to utilize them.

Six out of eleven practitioners did not define all types at the beginning. This is a possible consequence of them not knowing beforehand what notation they are going to use. They concentrated on sketching the problem and came up with notations as needed on the fly. They did not know whether they will re-use certain symbols after drawing them for the first time and therefore delayed the type definition to a later point during the experiment. The fact that our tool allows this is one of its main strengths. The behavior of deferring type definitions contrasts the results of our first study (Section 6.3). When a modeler sketches for herself, she does not need to coordinate or explain to anyone what she is doing. This might foster intentional ambiguity in sketches and could be a possible explanation why type definitions were deferred more in the second study.

One of the questions in our semi-structured interviews concerned possible usage scenarios for our tool. A common answer from the RE practitioners was that they would use the mobile version of FlexiSketch on-site with customers. Sometimes they would also take a photograph of the important machines or surroundings and include it in the sketch. They would then send the sketches consisting of structured information to a coworker. The coworker would further augment and annotate the sketches and send them back for another iteration. Like this, the sketches would evolve

into more concrete models over time. As the primary purpose of sketches in this usage scenario is communication, some of the RE practitioners stated that it is not important to define cardinalities. They do not feel the need to formalize their FlexiSketch sketches to an extent where cardinality definitions for the metamodel become important.

In contrast, the practitioners who focus more on software development said in the semi-structured interview that they like the ability to define cardinalities. Cardinalities are needed when models should be processed formally (e.g., for model transformations). In addition, some of them stated that they would like to have even more metamodeling options (such as, e.g., the option to define subtypes of types, and the option to enter complex constraints by using something similar to OCL – the object constraint language).

Seven out of eleven practitioners in our study had no problems to define cardinalities correctly. As long as our tool is used by RE practitioners (our target users), our study suggests that defining cardinalities can remain an optional feature. If, however, our focus would shift to model-driven engineering one day, we should provide additional guidance for defining cardinalities and include further metamodeling options.

6.5.3 Strengths and Weaknesses of FlexiSketch

Based on the analysis of the studies and the feedback from participants, we identified the following strengths and weaknesses of our tool.

Overall, our tool provides a lightweight metamodeling method that is easy to use by RE practitioners, at the expense that it is limited to node-and-edge structures and therefore supports a limited set of modeling languages. However, the flexibility and simplicity have been proven to be a major factor for acceptance of the tool. Also, results of an earlier study have shown that node-and-edge diagrams are among the most frequently used diagram types in early requirements elicitation [WSG13a]. A big advantage is that sketching and metamodeling activities can be performed at any time. The tool fosters incremental notation definitions.

The drag and drop mechanism can motivate users to perform some lightweight metamodeling (i.e., define types) in order to be able to re-use the types. This re-use can lead to consistent notations. It can also help novice modelers to provide meaningful type names. Furthermore, defining types can be seen as a form of documenting a sketch, which can lead to better understandability. As a tradeoff, users might prematurely reduce intended ambiguity in their sketches because they have to enter type names for using the drag and drop mechanism.

Less experienced modelers had difficulties to define cardinalities correctly in our study. It seems that the guidance for cardinality definitions currently provided by the tool is not enough. Also, the wizard could be improved in that regard. From a general perspective, it should be noted that our tool does not include any guidelines or mechanisms for assessing or improving the quality of a modeling language and its metamodel. That is because the tool is not meant to be a metamodeling tool per se, but is meant as a

sketching tool that allows users to formalize and export sketches in a structured way.

Regarding group work, our first study has shown that the collaborative mode of our tool allows users to simultaneously edit small portions of the sketch canvas. This would be difficult on a whiteboard due to the fact that each user needs a certain amount of physical space, but becomes possible if each user has its own input device. Furthermore, our collaboration setting motivated participants to take actively part in defining a modeling language collaboratively, rather than to hand off language definitions to a single person.

Initially, we expanded on the principle of flexibility by making FlexiSketch a mobile tool. While the advantage of mobility has been recognized by the study participants, the tool has been criticized for its small screen size. This aspect became more important in the group setting: when sketches become larger, users start to scroll and zoom, and they are more likely to lose the overview of what other users are doing. To minimize the issue of a small screen size, we have developed a desktop version of the tool. The desktop version shows an overview of the whole sketch canvas during collaborative work. However, our first study has shown that less experienced modelers have difficulties to cope with multiple screens at the same time. This problem can be avoided when users work collaboratively on an electronic whiteboard, using the desktop version of our tool. But, for better supporting collaborative work with the mobile tool, additional user awareness features need to be incorporated.

6.5.4 Threats to Validity

In this section we briefly discuss threats to validity for the two studies we had conducted.

Study 1: Simulated Workshops

Conclusion validity. The first study was performed with three student groups and three practitioner groups to get qualitative results and an in-depth understanding how groups create ad-hoc modeling notations. The small amount of six data points is a threat to conclusion validity. A complementary, quantitative study is necessary to mitigate this threat.

Internal validity. Participants were unfamiliar with the tool, and we gave an introduction to mitigate this threat. Yet, the desire of the participants to explore the new technology could have influenced the collaboration task. For example, it could have fostered simultaneous editing and added to the result that multiple group members participated in defining types. Furthermore, minor usability issues were potential distractions and could have influenced the collaboration task.

When participants know the researchers personally, they might be tempted to give overly positive feedback. To mitigate this threat, we asked students to fill out an anonymous online survey after the lecture. Participants of the two practitioner groups PG2 and PG3 did not know the researcher who performed the study with them,

which further mitigates this threat. We did not find a discrepancy between the feedback from PG1 and the other two practitioner groups.

Construct validity. Student groups had to create predefined diagram types. This is a possible threat because it can influence the amount of discussion needed about semantics (obviously, student groups talked less about semantics compared to practitioners groups). It can also minimize tool usability issues, since we already knew that these diagrams can be built with our tool. However, some of the results, such as the lack of micro-coordination in student groups, are independent of a particular modeling notation and therefore not affected by this threat. Furthermore, students had to solve a task that was constructed by us, while practitioners could work on a real-world task from their company. The participants might thus have had different levels of intrinsic motivation.

We told the groups that all elements should have types assigned at the end. This could have influenced their typing behavior, and therefore might have contributed to the result of consistent notations. However, we wanted to make sure that the groups define enough types (or define types at all), such that we can draw conclusions about their language definition behavior.

External validity. The limited number of students and practitioners who were involved in our evaluation activities, as well as the limited geographical distribution (Switzerland and Austria) is a known threat (convenience sampling according to proximity). However, we involved both novice and expert modelers with different backgrounds and skills. During the twenty-minute sessions, we

identified collaboration patterns that confirm the usefulness of our FlexiSketch approach for early requirements modeling. Whether the results can be generalized to longer sessions has yet to be verified. The behavior of modelers might differ over time.

Study 2: Quantitative Experiment

Conclusion validity. In our second study, the limited number of eleven practitioners involved could be seen as threat. However, we will continue to perform reality checks with practitioners, gathering additional data for the statements made in this article. Also, results from a previous experiment [WSG13a] do not contradict the latest findings. The student experiment was of a quantitative nature. It was performed at two Swiss universities, and thus the locality could be seen as threat.

Internal validity. Participants stated they were familiar with tablets or touch devices in general. However, they were using FlexiSketch for the first time. We included a tutorial at the beginning of the experiment to mitigate the threat to internal validity. However, we expect that involving trained FlexiSketch users would have led to better results. Furthermore, the student experiment took place as part of a lecture, which could have influenced students' motivation and therefore the quality of the results. Also, differences between the quality of class diagram and use case diagram solutions (including the metamodel) could be an effect of differences between the students of the two universities. All students were at the beginning of learning to model UML. At

the time of the experiment, there was no UML model type that was known by students from both universities. Thus, we could not mix the students from the two universities.

Regarding the differences between FlexiSketch solutions and paper solutions, it has to be noted that the different versions of the handouts could have affected the outcome. For example, presenting the modeling task, the type definition task, and the cardinality definition task as individual parts in the handout for the non-FlexiSketch version could have been too much of a help for those students. We decided to do this in order to have an equivalent help compared to the tool's wizard. However, many students who performed their task with the tool did not consult the wizard. This could explain why paper solutions outperformed tool solutions. Furthermore, the tool's restriction of permitting cardinality definitions only for relations that have already been fully typed was perceived as an usability problem by some students.

Another issue is that we used standard modeling notations within the student experiment. While our approach is rather meant to be used with custom languages, this allowed us to precisely measure completeness and correctness against a given ground truth, and therefore measure the metamodeling capabilities of novice modelers when supported by our tool.

Construct validity. When planning the experiment, we tried to avoid several threats to construct validity. For example, we conducted pilot tests with graduate students to verify that our handouts are understandable and that participants have enough

time for the main modeling task without suffering from time pressure. In the student experiment, students were randomly assigned to either work on the tasks with a tablet, or to be part of the control group and work on paper. This allowed us to compare tool solutions with solutions on paper, and therefore measure whether our tool itself biased the results.

Another possible threat is the absence of an objective ground truth against which we could have evaluated the results from practitioners. Metamodel completeness could be deduced from the model sketches, but for assessing metamodel correctness, we partly had to rely on practitioners' opinions since the used languages only existed in practitioners' minds.

In order to be able to compare student and practitioner solutions in terms of quality, we tried to give the same instructions in both experiments (apart from having a predetermined model type and task in the student experiment). For example, we mentioned the existence and functionality of the tool wizard in the same way. We only forced practitioners to use the wizard once they were done with the task (such that we could compare their pre-wizard results with student results). However, the students received their task as written instructions, while we told the instructions to practitioners orally.

External validity. In order to reduce threats, we evaluated FlexiSketch with both experienced modelers and modeling novices. We included different diagram types, both custom and standardized notations, to increase the generalizability of the experiment results.

6.6 Related Work

6.6.1 Sketching in SE

Sketching is an important method to foster creativity and discuss design ideas [CGH08, GD96, VdL02]. Researchers have identified the need for sketching in software engineering to support creativity and idea generation a long time ago [Sut63, EHS69]. Various studies investigate the reasons for the popularity of whiteboards and sketching in SE [Goe95, Tve02, VdL02, CVDK07].

As a result, there exist many approaches that augment formal modeling tools with sketch recognition, for example, SUMLOW [CGH08], Tahuti [HD06], SketchREAD [AD04], and Scribble [SA13]. A detailed overview is provided by Johnson et al. [JGHYLD09]. Furthermore, many sketch-recognition based approaches focus on user interface design, e.g., [CSVV07] and [LM01]. These approaches have in common that they start from predefined modeling notations, and then provide a sketch interface capable of recognizing these notations. Therefore, the tools cannot interpret any drawings which do not conform to the given notations. In contrast, our approach starts on the informal side, mimicking paper, and permits the use of arbitrary notations.

On the informal end of the spectrum, tools such as Calico [MBD+10] and Sketch for Eclipse [SB10] support informal sketching with the possibility to structure the information, but they do not provide further formalization capabilities. AugIR [KHG15] is a sketching

tool running on multiple electronic whiteboards, and enables users to add multiple annotations to sketched elements. In contrast to our work, their goal is not to distill metamodel information for a formalization of the sketches, but they use the annotations to automatically link related concepts found in other sketches to the annotated elements. This allows users to easily navigate between related sketches and artifacts. BITKit [OBS+10] is an approach that, similar to FlexiSketch, allows users to assign types to geometrical shapes. However, BITkit does not contain sketching, and the authors do not discuss how the diagrams can be formalized, or how an underlying metamodel can be created. InkKit [PF07] allows to add additional diagram types including recognition and export support, but uses DLL plugins for this purpose.

6.6.2 End-User Metamodeling

Metamodeling tools such as MetaBuilder [FHH00] or MetaEdit+ [KLR96] provide graphical metamodel editors for the creation of modeling languages and editors. DiaMeta [Min06] and MaramaSketch [GH07] allow to develop diagram editors which support free-hand editing. However, with these tools, users cannot create custom modeling languages by example, but must define them beforehand in the metamodeling tool. Once the editor tools are compiled, languages cannot be extended directly in these editors. In contrast, our flexible metamodeling approach uses metamodeling by example. This also allows for an iterative metamodel creation process where business stakeholders can draw models before the metamodel is complete. Similarly, the Electronic Cocktail

Napkin [GD96] provides a sketching interface and allows users to define constructs later on in order to resolve ambiguity and vagueness. However, the tool is not tailored to the SE domain and requires scripting or programming knowledge to create meta-descriptions.

The operation of metamodeling tools usually requires good metamodeling knowledge. In contrast, there are not many approaches for end-user metamodeling (i.e., non metamodeling experts). One reason is that, from a metamodeling perspective, it was long believed that metamodeling should only be done by metamodeling experts [Kle08]. Indeed, it has been shown that end-user metamodeling is hard to achieve (e.g, see [QGW10]). In contrast, we concentrate on lightweight metamodeling (or just enough metamodeling) by example for creating ad-hoc notations in an end-user friendly way (e.g., for requirements engineers and domain experts). Our approach considers the results of Qattos et al. [QGW10], who report on experiments showing that metamodeling by example (seeing concrete graphical examples) results in better metamodels than a wizard-based method. Furthermore, Cho et al. [CGS12] discuss challenges of metamodeling by example, and Kuhrmann [Kuh11] argues for the necessity of user assistance for DSML creation. One of the challenges lies in the co-evolution of models and metamodels. Co-evolution issues are also discussed in [CREP08, DRIP13, SCDLG12, Wac07]. Since our main goal is to formalize model sketches, we have neglected the co-evolution topic so far by having one metamodel per model sketch, and always updating the metamodel according to the sketch. But co-evolution should be a topic for future work.

Publications about the evaluation of end-user metamodeling are still scarce. For example, Gabrysiak et al. [GGLS11] present different approaches to create metamodels before or after creating models in their position paper, but do not provide an evaluation. Volz and Jablonski [VZJ11] propose an approach that allows the step-wise formalization of sketched models. Cuadrado et al. [SCDLG12] and López-Fernández et al. [LFCGL15] propose bottom-up metamodeling, similar to our approach, but they use separate tools for the modeling and metamodeling activities. No user evaluation is provided in any of these papers. Wouters [Wou13] proposes a notation-driven approach to create a metamodel, which is supposed to help in creating notations that match domain experts' expectations. The work is evaluated in an industrial case study and an empirical study. However, the approach uses a graphical grammar instead of free sketching. Another thread of research investigates data mining technology for automatic creation of metamodels from a large set of given example models, e.g., MARS [JMGB08, MHB+09]. In contrast, FlexiSketch aims at building metamodels for a small set of sketches or individual sketches, so approaches requiring large data sets are not applicable in our case.

6.6.3 Collaborative Design and Language Creation

In requirements and software engineering, collaboration is often researched in the context of design [BGCB10, LMLvdH13], and user

interface creation [JA07, LMLvdH13, SBV12]. Some researchers focus on understanding the behavior and low-level collaboration patterns of participants when working with physical media, e.g., [GG00, Tan91], which resulted in design guidelines for software tools that support collaborative work [GG98, GG00, HLS+10, Tan91]. Today, there are many software tools that support collaborative sketching and design work (e.g., Calico [MBD+10], The NiCE Discussion Room [HLS+10]). Settings with such digital tools have the potential to change the way how engineers and designers work and collaborate. Users might show different collaboration behaviors if they work with digital tools instead of physical media, because, e.g., workspace awareness can be different. Therefore, when introducing a new software tool, it makes sense to study the influence of this tool on collaboration and sketching behavior. Examples of such studies can be found in, e.g., [HLS+10, LW08, MBD+10, MLPvdH14].

While we also looked at collaborative sketching behavior when using FlexiSketch, the main focus of our first study was to investigate how requirements engineers collaboratively design, agree on, and define notations. Related work on this subject is still scarce. Dekel and Herbsleb [DH07] performed an observational study to find out what kind of notations are used in object-oriented design, and how they evolve during sessions. Ossher et al. [OJDB10] investigated notations used in software design sessions to conclude whether their flexible modeling approach can provide appropriate support. Both works used physical media in the studies. In contrast, our study investigates how non-expert metamodelers choose and define notations when using a flexible software tool.

6.7 Conclusions

As a part of our flexible modeling approach, we developed the FlexiSketch tool. The multi-screen, node-and-edge diagram sketching tool allows users to define custom notations on the fly by assigning types to elements and specifying cardinality rules. In this work, we presented two studies about lightweight metamodeling using our tool: a qualitative study about how requirements engineers sketch and define ad-hoc notations collaboratively, and a quantitative study about how well modeling novices can handle the metamodeling mechanisms. The qualitative study indicates that the tool fosters interleaving of sketching and type-defining activities, and motivates all members of a group to perform both activities. Users managed to define consistent notations for their sketches collaboratively and reached a common understanding of the respective notations. The quantitative study shows that novice modelers with no metamodeling knowledge can define types on a metalevel, but have trouble in correctly defining cardinality rules and sometimes think on the model level instead of the metamodel level.

Results from the first study also suggest that having additional awareness features in the tool (for knowing what the other users are doing) would be beneficial. Furthermore, the second study shows that active guidance by the tool is needed if novice modelers shall be able to provide correct cardinality rules. One option could be to show concrete examples as done in [QGW10]. We also saw that some users are only willing to provide meta-information if

they receive immediate benefits, e.g., by re-using types via drag and drop. While not all practitioners would want to formalize the sketches created with our tool, they understand the importance of providing meta-information to allow the formalization of model sketches such that they can be exported and re-used in other tools. In our future work, we plan to improve FlexiSketch according to these results. This includes the integration of additional user awareness features for the collaborative version, tool guidance for cardinality definitions, and usability improvements. We also plan to perform longitudinal evaluations in real-world software projects, and investigate how sketches made with our tool are re-used and changed during the projects. This will allow us to gather feedback about the quality of sketches and metamodels from people who will actually have to re-use these artifacts as a part of their work.

Acknowledgment

We would like to thank André van der Hoek for his very valuable feedback and comments on an earlier version of this article.

Chapter 7

Conclusion

7.1 Thesis Summary and Contribution

Even with the wide variety of software modeling tools that are available today, software and requirements engineers often use whiteboards, paper, or office tools to sketch models and ideas, especially in creative requirements elicitation and early design sessions. The problem with such sketches is that they cannot be re-used easily. They are cumbersome to change, and software modeling tools cannot derive the meanings of the sketched models, because from a technical viewpoint these models are just images without any underlying syntax or semantics. Therefore, engineers have to manually create more formal models with the information contained in the sketches if they want to formalize and re-use that information.

In this thesis, we proposed a new flexible modeling approach for a more seamless integration of sketched models into the overall requirements engineering process. The core idea of our tool-supported approach is that its users can switch between sketching/modeling and metamodeling activities at any time. To prove the concept of our approach, we developed the FlexiSketch tool. The tool is meant as an alternative to paper and whiteboards and mimics those physical drawing media by displaying a white sketch canvas and a minimalistic user interface. Our approach includes free-form drawing, focuses on models that consist of symbols and links, and allows for annotating the sketched elements with types and cardinality rules. The annotation functionality provides the option to step-wise formalize model sketches. We generate a simple metamodel semi-automatically by analyzing the sketched elements and the user annotations. This allows for exporting a sketched model and its metamodel as XML files, and – by providing respective parsers – enables users to re-use these documents in other software modeling and metamodeling tools.

Our research methodology is based on the engineering cycle presented in [WH06]. Following an iterative process, we performed several qualitative and quantitative studies to evaluate our approach, and we refined our tool according to the results. For example, we developed a desktop version of the tool that runs on electronic whiteboards, and we added collaboration features to the mobile tool.

We conclude that, with our approach, the process of formalizing sketched models is no longer a manual task, but a tool-supported,

partially automated activity. The step-wise formalization options provided by our approach reflect an iterative, evolutionary development of models. Also, sketched models created with our tool-supported approach are easier to re-use compared to models created on physical media. Moreover, our approach can not only be used to create model sketches that are amenable to further processing, but also to create simple custom modeling languages. To our knowledge, our approach is unique in the sense that it provides collaborative lightweight metamodeling capabilities in a sketching environment.

7.2 Revisiting the Research Questions

This section summarizes how we answer this thesis' research questions from Section 1.3.

RQ 1: How do requirements engineers re-use information contained in informal model sketches? In Chapter 2, we present the results of a qualitative interview with nine SE and RE practitioners from different companies. The results are consistent with the outcome of a literature review about the use of sketches and physical media in software engineering processes (see, e.g., [CVDK07] and [WHD+11]). The interview answers can be summarized into four kinds of re-use: i) engineers do not re-use some sketches, because they only remain valid for some weeks (either due to the fact that sketches on physical media are cumbersome to change, or because the contained information is obsolete),

ii) they take photographs of the sketches and include the pictures in other documents, iii) they take photographs to preserve the information for later when they re-create the models (or parts of them) in a software modeling tool, and iv) they create some other kind of documentation, or communicate the sketched content verbally. We used the outcome from the interview and the literature review to construct a tool-supported approach that allows for the export of created sketches in a semi-formal form. After we presented our approach to the practitioners and let them test our initial tool, they named concrete ways of how they would like to re-use sketches created with our tool. We report these results in Chapter 2.

RQ 2: What is an effective tool-supported approach to enhance the reusability of informal model sketches? In Chapter 2, we introduce our flexible modeling approach and the FlexiSketch tool. The tool analyzes sketched elements while the user is drawing, and the user can – on demand – define types as well as cardinality rules for links. This leads to a simple metamodel, and thus our approach enables the export of sketched models in formal and informal ways. Models can be exported as images or as XML files, and then be re-used in various ways. This solution supports and simplifies the re-use of sketched models in the ways we identified when answering RQ 1. Chapter 3 reports on an extension of our approach that takes the collaboration of multiple engineers into account. It discusses how sketches and simple modeling languages can be created collaboratively when using our approach. Our solution enables simultaneous editing of the same workspace with multiple devices. Chapter 4 provides an

in-depth discussion of the metamodeling mechanisms covered by our approach and explains how the metamodel is generated. Our interface design choices and the meta-metamodel we use in the tool are tailored for simple diagram sketches consisting of nodes and edges. Chapter 2 provides evidence that node-and-edge diagrams are among the most frequent model types found in sketches from software and requirements engineers.

RQ 3: How well can requirements engineers perform sketching/modeling with our approach? The usability and utility of our approach in regard to modeling (this excludes the metamodeling part) is discussed in the evaluation section of Chapter 2. We conducted a qualitative experiment with eight computer science students (undergraduate level and PhD level) and nine practitioners from the software and requirements engineering fields where they engaged in sketching and modeling tasks with our FlexiSketch tool. Over the years, we augmented and refined the tool according to the evaluation results. For example, we created a collaborative version and a desktop version of the tool. Overall, we found that our approach is successful in providing free-form sketching as well as an interface that enables easy iteration over model sketches. However, since the tool is a research prototype, its usability still leaves some space for improvements. While the evaluations reported in Chapters 5 and 6 focus on the metamodeling part of our approach, they also provide answers to this research question. This is because the nature of our approach does not allow us to evaluate its metamodeling part without performing sketching and modeling activities.

RQ 4: How well does our approach support the creation of a lightweight metamodel within a sketching environment? This question is split into the two sub-questions RQ 4.1 and RQ 4.2. We therefore answer this question by answering the two sub-questions.

RQ 4.1: How well can requirements engineers with varying degrees of modeling knowledge perform lightweight metamodeling with our approach? This question gets answered in the evaluation section of Chapter 6. We performed a quantitative study with a total of 107 undergraduate students from two different universities, complemented by a qualitative study with eleven SE and RE practitioners. We found that RE experts – our main target users – understand the metamodeling mechanisms of our approach and can correctly define their sketched model elements as well as specify cardinality constraints for relations between the elements. Answers from the practitioners suggest that our approach contains adequate metamodeling features for our main target users, while SE experts (software developers with good knowhow in object-oriented programming) would like to provide more complex metamodel information. The students in our study had less than one year of modeling experience at the time we conducted the study. They successfully assigned correct types to their sketched elements, but had problems to define cardinality constraints correctly (some of them did not understand that cardinalities apply to types of relations and not only to a specific instance of the relation type). One reason is that type definitions are much more tangible: types appear in a palette immediately and can be re-used via a drag and drop mechanism. This helps

a modeler to verify that she defined a type on the right level of abstraction. In contrast, defined cardinality rules remain invisible and do not affect the modeling task as long as the tool does not enforce compliance of a model to the underlying metamodel. Enforcing cardinality rules is an optional feature in our tool that is turned off by default. Interview answers from RE practitioners lead to the conclusion that they want to remain flexible while working with our tool, and that they do not care about the definition of cardinality rules while creating model sketches. In RE, models are often used as a means of communication; a formalization of the models up to an extent where cardinality rules for the metamodel become important is often out of scope in the daily work of RE practitioners. On the other side, if model sketches created with our tool should be processed in more formal ways (e.g., model transformations), we could add additional guidance for defining cardinalities. However, the persons who need this level of formality are typically very knowledgeable in modeling and metamodeling, and do not need this guidance.

RQ 4.2: How do requirements engineers define modeling languages collaboratively when using our approach? The answer to this question is given in the evaluation section of Chapter 5. We performed a qualitative study with three small groups of students and three small groups of practitioners. They had to fulfill a modeling task and were asked to have types assigned to all sketched elements at the end of the task. All groups made use of the ability to sketch and define a modeling notation collaboratively. From all 17 participants, only one participant did not perform any type definition. Types were defined during the whole modeling

task, not only at the beginning, i.e., the notations grew incrementally. Although every participant had his/her own touchscreen for editing, practitioners communicated and coordinated their actions well. In contrast, some students were lacking coordination because their attention was drawn to their individual screens. We conclude that we created a useful and adequate approach for collaborative metamodeling in a sketching environment. However, the study setting with multiple screens imposes high demands on the participants cognitive abilities to split their focus between their own screens and communicating with the other group members. In cases where this is a problem, it might be advisable to use FlexiSketch Desktop on an electronic whiteboard instead of individual devices.

With the help of the answers to the research questions, we can conclude that the thesis statement is indeed true: our approach combines informal model sketching with formalization mechanisms in a single (mobile) tool, and it enables requirements engineers to transform their sketches into semi-formal models (by defining types and cardinality rules). Models and metamodels can be exported as XML files and re-used in other tools such as MetaEdit+ (once they got parsed into a tool-compatible file format). However, answers to the research questions have also shown that there are opportunities to improve our approach. Moreover, as shown in the next section, further evaluations could be performed to assess the utility of our approach in more detail.

7.3 Next Steps

It is relatively difficult to thoroughly evaluate an approach such as ours. First, our approach is interactive and relies on human participation for evaluating its usability and utility. Thus it cannot be benchmarked just like an automated approach. We need persons that are willing to participate in the evaluations, and there will always be a human factor. Second, the real benefits of our approach appear when artifacts created with our tool are re-used. This means that it is not possible to evaluate our whole approach in a single session. Participants, especially practitioners from industry, must agree to test our tool over a longer amount of time and use it in multiple study sessions. Therefore, an important next step is to evaluate FlexiSketch in longitudinal studies where practitioners use the tool over a longer amount of time and also re-use sketches that they created earlier with the tool. In the end, only practitioners who have to re-use the produced artifacts will know whether the sketches created with our tool are indeed understandable and easy to re-use.

Practitioners who have to re-use artifacts created with FlexiSketch would also be able to make statements about the quality of the artifacts. This leads to more interesting future work: assessing the quality of the sketches and metamodels that are created with our tool. However, this is a complex task that requires more in-depth research. Related work provides some pointers where to start, e.g., [SNH+09, POB00, Jea13]. For evaluating the quality and reusability of sketches, a study could compare our tool with

physical media such as whiteboards and paper. Creating a good experiment design for this case can be difficult since our tool has features that go beyond sketching, and there is no counterpart for this when working with physical media.

Other future work could focus on the metamodeling part of our approach. The topic of lightweight metamodeling holds many possible research directions. Here we provide three examples: first, literature about collaborative metamodeling is still scarce, which could be changed by future work. Second, if our approach is to be used by novice modelers, or in an educational context, it would be valuable to investigate how more metamodeling guidance can be provided in an understandable and unobtrusive way. Third, the metamodeling part in our approach focuses on the concrete and abstract syntax, while computer-readable semantics are neglected. So far, semantics exist only implicitly in the form of the user-chosen type names. Semi-automatic approaches for creating and assigning semantics could be explored in future work. From the perspective of model-driven engineering, this would not only allow for the transformation of sketches into models, but would pave the way for source code generation.

Finally, research could be done to find out how our approach can be made even more flexible. On the one side, one could think about new export options (how to export information from FlexiSketch in different ways and formats). On the other side, one could investigate how our approach can support additional data structures: currently, our approach focuses on the formalization of diagrams consisting of nodes and edges. It would be interesting to

find out whether it is possible to include formalization capabilities for other types of sketches while preserving the tool's ease of use and without having an interface that gets too convoluted. Such other types of sketches could include sketches that use the spatial relations of elements to depict chronologies, hierarchies or containment relationships, sketches that contain tables and lists, etc.

The above list of possible future work is not comprehensive: we think that our new flexible modeling approach has the ability to generate many interesting research questions and opportunities for future work. Therefore, we do not see our approach just as a solution to an existing problem, but see it as something that has huge potential to promote and inspire more research on creative and flexible modeling support in order to make the lives of software and requirements engineers easier and more colorful.

Bibliography

[AD04] Christine Alvarado and Randall Davis. SketchREAD: a multi-domain sketch recognition engine. In *Proceedings of the 17th annual ACM symposium on User interface software and technology (UIST 2004)*, pages 23–32. ACM, 2004.

[AK03] Colin Atkinson and Thomas Kühne. Model-driven development: a metamodeling foundation. *IEEE Software*, 20(5):36–41, 2003.

[BCW12] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2012.

[BGCB10] Stacy Branham, Gene Golovchinsky, Scott Carter, and Jacob T. Biehl. Let's go from the whiteboard: supporting transitions in work through whiteboard

capture and reuse. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems (CHI 2010)*, pages 75–84. ACM, 2010.

[BM08] Florian Brieler and Mark Minas. Recognition and processing of hand-drawn diagrams using syntactic and semantic analysis. In *Proceedings of the 2008 Working Conference on Advanced Visual Interfaces (AVI)*, pages 181–188. ACM, 2008.

[Bro04] Alan W. Brown. Model driven architecture: Principles and practice. *Software and Systems Modeling*, 3(4):314–327, 2004.

[BZS+11] Nelson Baloian, Gustavo Zurita, Flávia Maria Santoro, Renata Mendes Araujo, Sean Wolfgan, Douglas Machado, and José A. Pino. A collaborative mobile approach for business process elicitation. In *Proceedings of the 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2011)*, pages 473–480, June 2011.

[CC09] Brock Craft and Paul Cairns. Sketching sketching: Outlines of a collaborative design method. In *Proceedings of the 23rd British HCI Group Annual Conference on People and Computers: Celebrating People and Technology (BCS-HCI 2009)*, pages 65–72. British Computer Society, 2009.

[CG11] Hyun Cho and Jeff Gray. Design patterns for metamodels. In *Proceedings of the Compilation of the Co-located Workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11 (SPLASH 2011 Workshops)*, pages 25–32. ACM, 2011.

[CGH08] Qi Chen, John Grundy, and John Hosking. SUMLOW: early design-stage sketching of UML diagrams on an e-whiteboard. *Software – Practice and Experience*, 38(9):961–994, 2008.

[CGS12] Hyun Cho, J. Gray, and E. Syriani. Creating visual domain-specific modeling languages from end-user demonstration. In *2012 ICSE Workshop on Modeling in Software Engineering (MISE)*, pages 22–28, June 2012.

[CREP08] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. Automating coevolution in model-driven engineering. In *Proceedings of the 12th International IEEE Enterprise Distributed Object Computing Conference (EDOC 2008)*, pages 222–231. IEEE Computer Society, 2008.

[CSVV07] Adrien Coyette, Sascha Schimke, Jean Vanderdonckt, and Claus Vielhauer. Trainable sketch recognizer for graphical user interface design. In Cécilia Baranauskas, Philippe Palanque, Julio Abascal, and

Simone Diniz Junqueira Barbosa, (Eds.): *Human-Computer Interaction – INTERACT 2007*, volume 4662 of *Lecture Notes in Computer Science*, pages 124–135. Springer Berlin Heidelberg. 2007.

[CVDK07] Mauro Cherubini, Gina Venolia, Rob DeLine, and Andrew J. Ko. Let's go to the whiteboard: how and why software developers use drawings. In *Proceedings of the 25th International Conference on Human Factors in Computing Systems (CHI 2007)*, pages 557–566. ACM, 2007.

[DCC+12] Quoc Do, Stephen Cook, Peter Campbell, William Scott, Kevin Robinson, Wayne Power, and Despina Tramoundanis. Requirements for a metamodel to facilitate knowledge sharing between project stakeholders. *Procedia Computer Science*, 8:285 – 292, 2012.

[DCJ11] James R. Douglass, Nicholas Chen, and Ralph E. Johnson. The Language of Languages research project: Unifying concepts expressed across different notations. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion (OOPSLA 2011)*, pages 15–16. ACM, 2011.

[DH07] Uri Dekel and James D. Herbsleb. Notation and representation in collaborative object-oriented de-

sign: An observational study. *SIGPLAN Notices*, 42(10):261–280, October 2007.

[DHT00] Christian Heide Damm, Klaus Marius Hansen, and Michael Thomsen. Tool support for cooperative object-oriented design: Gesture based modelling on an electronic whiteboard. In *Proceedings of the 2000 International Conference on Human Factors in Computing Systems (CHI)*, pages 518–525. ACM, 2000.

[DRIP13] Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. Managing the coupled evolution of metamodels and textual concrete syntax specifications. In *Proceedings of the 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2013)*, pages 114–121, 2013.

[EHS69] Thomas O. Ellis, John F. Heafner, and William L. Sibley. The GRAIL project: An experiment in man-machine communications. Technical Report RAND Memorandum RM-5999-ARPA, RAND Corporation, 1969.

[EP11] Martin J. Eppler and Roland Pfister. Sketching as a tool for knowledge management: An interdisciplinary literature review on its benefits. In *Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies (i-KNOW 2011)*. Article No. 11. ACM, 2011.

[ESSD08] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, (Eds.): *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer London, 2008.

[FHH00] Ian Ferguson, Andrew Hunter, and Colin J. Hardy. Metabuilder: The diagrammer's diagrammer. In *Theory and Application of Diagrams*. Volume 1889 of *Lecture Notes in Computer Science*, pages 407–421. Springer, 2000.

[Fin94] Piotr Findeisen. The Metaview system. Technical report, University of Alberta, Alberta, 1994.

[FR07] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *Proceedings of the 2007 Future of Software Engineering Symposium (FOSE)*, pages 37–54. IEEE Computer Society, 2007.

[GD96] Mark D. Gross and Ellen Yi-Luen Do. Ambiguous intentions: a paper-like interface for creative design. In *Proceedings of the 9th annual ACM Symposium on User Interface Software and Technology (UIST 1996)*, pages 183–192. ACM, 1996.

[GG98] Carl Gutwin and Saul Greenberg. Design for individuals, design for groups: Tradeoffs between power

and workspace awareness. In *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work (CSCW)*, pages 207–216. ACM, 1998.

[GG00] Carl Gutwin and Saul Greenberg. The mechanics of collaboration: Developing low cost usability evaluation methods for shared workspaces. In *Proceedings of the 9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2000)*, pages 98–103. IEEE, 2000.

[GGLS11] Gregor Gabrysiak, Holger Giese, Alexander Lüders, and Andreas Seibel. How can metamodels be used flexibly? In *ICSE Workshop on Flexible Modeling Tools*, 2011. Available online [last checked 07/04/16]: `http://www.researchgate.net/publication/235675453_How_Can_Metamodels_Be_Used_Flexibly`.

[GH07] John Grundy and John Hosking. Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool. In *Proceedings of the 29th International Conference on Software Engineering (ICSE 2007)*, pages 282–291. IEEE Computer Society, 2007.

[GHL+13] John Grundy, John Hosking, Karen Li, Norhayati Mohd Ali, Jun Huh, and Richard Lei Li. Generating domain-specific visual language tools from abstract visual specifications. *IEEE Transactions on Software Engineering*, 39(4):487–515, April 2013.

[GJPR10] Florian Geyer, Hans-Christian Jetter, Ulrike Pfeil, and Harald Reiterer. Collaborative sketching with distributed displays and multimodal interfaces. In *Proceedings of the 5th ACM International Conference on Interactive Tabletops and Surfaces (ITS 2010)*, pages 259–260. ACM, 2010.

[GM94] Saul Greenberg and David Marwood. Real time groupware as a distributed system: Concurrency control and its effect on the interfaces. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work (CSCW)*, pages 207–217. ACM, 1994.

[Goe95] Vinod Goel. *Sketches of Thought*. The MIT Press, Cambridge, MA, 1995.

[GSFV14] Paola Gómez, Mario Sánchez, Hector Florez, and Jorge Villalobos. An approach to the co-creation of models and metamodels in enterprise architecture projects. *Journal of Object Technology*, 13(3):2:1–29, July 2014.

[GW07] Martin Glinz and Roel J. Wieringa. Guest editors' introduction: Stakeholders in requirements engineering. *IEEE Software*, 24(2):18–20, 2007.

[HD06] Tracy Hammond and Randall Davis. Tahuti: a geometrical sketch recognition system for UML class diagrams. In *ACM SIGGRAPH 2006 Courses, Article No. 25, ACM*, 2006.

[HHL+07] Joshua Hailpern, Erik Hinterbichler, Caryn Leppert, Damon Cook, and Brian P. Bailey. TEAM STORM: Demonstrating an interaction model for working with multiple ideas during creative group work. In *Proceedings of the 6th ACM SIGCHI Conference on Creativity & cognition (C&C 2007)*, pages 193–202. ACM, 2007.

[HLS+10] Michael Haller, Jakob Leitner, Thomas Seifried, James R. Wallace, Stacey D. Scott, Christoph Richter, Peter Brandl, Adam Gokcezade, and Seth Hunter. The NiCE Discussion Room: Integrating paper and digital media to support co-located group meetings. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems (CHI 2010)*, pages 609–618. ACM, 2010.

[Hut95] Edwin Hutchins. *Cognition in the Wild*. MIT Press, Cambridge, MA, USA, 1995.

[ICLF+13] Javier Luis Cánovas Izquierdo, Jordi Cabot, Jesús J. López-Fernández, Jesús Sánchez Cuadrado, Esther Guerra, and Juan de Lara. Engaging end-users in the collaborative development of domain-specific modelling languages. In *Proceedings of the 10th International Conference on Cooperative Design, Visualization, and Engineering (CDVE 2013)*. Volume 8091 of *Lecture Notes in Computer Science*, pages 101–110. Springer, 2013.

[JA07] Maria Johansson and Mattias Arvola. A case study of how user interface sketches, scenarios and computer prototypes structure stakeholder meetings. In *Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI... but not as we know it (BCS-HCI 2007)*, pages 177–184. British Computer Society, 2007.

[Jea13] Cédric Jeanneret. *Measuring abstraction with footprinting: confronting software models with their purposes*. PhD thesis, University of Zurich, 2013.

[JGHYLD09] Gabe Johnson, Mark D. Gross, Jason Hong, and Ellen Yi-Luen Do. Computational support for sketching in design: A review. *Foundation and Trends in Human-Computer Interaction*, 2(1):1–93, January 2009.

[JMGB08] Faizan Javed, Marjan Mernik, Jeff Gray, and Barrett R. Bryant. Mars: A metamodel recovery system using grammar inference. *Information and Software Technology*, 50(9-10):948–968, August 2008.

[KHG15] Markus Kleffmann, Marc Hesenius, and Volker Gruhn. Connecting UI and business processes in a collaborative sketching environment. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2015)*, pages 200–205. ACM, 2015.

[KHK11] Heiko Kern, Axel Hummel, and Stefan Kühne. Towards a comparative analysis of meta-metamodels. In *Proceedings of the Compilation of the Co-located Workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11 (SPLASH 2011 Workshops)*, pages 7–12. ACM, 2011.

[KKP+09] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. Design Guidelines for Domain Specific Languages. In *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM 2009)*. TR no B-108, Helsinki School of Economis, Florida, USA, 2009.

[Kle08] Anneke Kleppe. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Addison-Wesley Professional, 2008.

[KLR96] Steven Kelly, Kalle Lyytinen, and Matti Rossi. MetaEdit+: A fully configurable multi-user and multi-tool CASE and CAME environment. In Panos Constantopoulos, John Mylopoulos, and Yannis Vassiliou, (Eds.): *Advanced Information Systems Engineering*, volume 1080 of *Lecture Notes in Computer Science*, pages 1–21. Springer Berlin / Heidelberg, 1996.

[Kuh11] Marco Kuhrmann. User assistance during domain-specific language design. In *ICSE Workshop on*

Flexible Modeling Tools, 2011. Available online [last checked 07/04/16]: `https://www4.in.tum.de/publ/papers/kuhrmann2011a.pdf`.

[Lew95] James R. Lewis. IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction*, 7(1):57–78, January 1995.

[LFCGL15] Jesús J. López-Fernández, Jesús Sánchez Cuadrado, Esther Guerra, and Juan Lara. Example-driven meta-model development. *Softw. Syst. Model.*, 14(4):1323–1347, October 2015.

[LLO13] Remo Lemma, Michele Lanza, and Fernando Olivero. CEL: modeling everywhere. In *Proceedings of the 35th International Conference on Software Engineering (ICSE 2013)*, pages 1323–1326. IEEE Press, 2013.

[LM01] James A. Landay and Brad A. Myers. Sketching interfaces: Toward more human interface design. *Computer*, 34(3):56–64, 2001.

[LMLvdH13] Dastyni Loksa, Nicolas Mangano, Thomas D. LaToza, and André van der Hoek. Enabling a classroom design studio with a collaborative sketch design tool. In *Proceedings of the 35th International Conference on Software Engineering (ICSE 2013)*, pages 1073–1082. IEEE, 2013.

[LTH12] Joon Suk Lee, Deborah Tatar, and Steve Harrison. Micro-coordination: Because we did not already learn everything we need to know about working with others in kindergarten. In *Proceedings of the 2012 ACM Conference on Computer Supported Cooperative Work (CSCW)*, pages 1135–1144. ACM, 2012.

[LW08] Lai-Chung Lee and Whei-Jane Wei. Comparison study on sketching behaviors with and without interactive pen displays. In *Proceedings of the 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2007)*, pages 567–574. Springer-Verlag, 2008.

[MBD+10] Nicolas Mangano, Alex Baker, Mitch Dempsey, Emily Navarro, and André van der Hoek. Software design sketching with Calico. In *Proceedings of the 2010 IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 23–32. ACM, 2010.

[MHB+09] Marjan Mernik, Dejan Hrncic, Barrett R. Bryant, Alan P. Sprague, Jeffrey Gray, Qichao Liu, and Faizan Javed. Grammar inference algorithms and applications in software engineering. In *Proceedings of the 22th International Symposium on Information, Communication and Automation Technologies (ICAT 2009)*, pages 1–7. IEEE, 2009.

[Min06] Mark Minas. Generating meta-model-based freehand editors. In *Proceedings of the 3rd International Workshop on Graph Based Tools (GraBaTs 2006), Satellite event of the 3rd International Conference on Graph Transformation*, volume 1 of *Electronic Communications of the EASST*, 2006.

[MLPvdH14] Nicolas Mangano, Thomas D. LaToza, Marian Petre, and André van der Hoek. Supporting informal design with interactive whiteboards. In *Proceedings of the 32nd International Conference on Human Factors in Computing Systems (CHI 2014)*, pages 331–340. ACM, 2014.

[Moo03] Daniel L. Moody. The method evaluation model: a theoretical model for validating information systems design methods. In *Proceedings of the 11th European Conference on Information Systems (ECIS 2003)*, pages 1327–1336, 2003.

[Moo10] Daniel L. Moody. The "physics" of notations: A scientific approach to designing visual notations in software engineering. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010)*, volume 2, pages 485–486. ACM, 2010.

[MOS+11] Nicolas Mangano, Harold Ossher, Ian Simmonds, Matthew Callery, Michael Desmond, and Sophia

Krasikov. Blending freeform and managed information in tables (NIER Track). In *Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011)*, pages 840–843. ACM, 2011.

[MSBS03] Daniel L. Moody, Guttorm Sindre, Terje Brasethvik, and Arne Sølvberg. Evaluating the quality of information models: empirical testing of a conceptual model quality framework. In *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003)*, pages 295–305. IEEE Computer Society, 2003.

[OBS+10] Harold Ossher, Rachel Bellamy, Ian Simmonds, David Amid, Ateret Anaby-Tavor, Matthew Callery, Michael Desmond, Jacqueline de Vries, Amit Fisher, and Sophia Krasikov. Flexible modeling tools for pre-requirements analysis: conceptual architecture and research challenges. In *Proceedings of the 2010 ACM International Conference on Object-Oriented Programming Systems Languages and Applications (OOPSLA)*, pages 848–864. ACM, 2010.

[OJDB10] Harold Ossher, Bonnie John, Michael Desmond, and Rachel Bellamy. Are flexible modeling tools applicable to software design discussions? IBM Tech. Rep. RC24949 (W1002-054), 2010.

[OvdHS+10] Harold Ossher, André van der Hoek, M.-A. Storey, J. Grundy, and Rachel Bellamy. Workshop on flexible modeling tools (FlexiTools). In *Proceedings of*

the 32nd International Conference on Software Engineering (ICSE 2010), pages 441–442. ACM, 2010.

[PA04] Beryl Plimmer and Mark Apperley. INTERACTING with sketched interface designs: an evaluation study. In *Proceedings of the 2004 CHI Extended Abstracts on Human Factors in Computing Systems (CHI EA)*, pages 1337–1340. ACM, 2004.

[PF07] Beryl Plimmer and Isaac Freeman. A toolkit approach to sketched diagram recognition. In *Proceedings of the 21st British HCI Group Annual Conference on People and Computers (BCS-HCI 2007)*, pages 205–213. British Computer Society, 2007.

[PGS+12] Florian Perteneder, Christian Grossauer, Thomas Seifried, Jagoda Walny, John Brosz, Tony Tang, and Sheelagh Carpendale. Idea Playground: When brainstorming is not enough. In *Designing Collaborative Interactive Spaces for e-Creativity, e-Science and e-Learning Workshop (AVI)*, 2012. Available online [last checked 12/05/15]: `http://hci.uni-konstanz.de/downloads/dcis2012_Perteneder.pdf`.

[PMMH93] Elin Rønby Pedersen, Kim McCall, Thomas P. Moran, and Frank G. Halasz. Tivoli: An electronic whiteboard for informal workgroup meetings. In *Proceedings of the 1993 INTERACT and CHI Conference on Human Factors in Computing*, pages 391–398. ACM, 1993.

[POB00] Richard F. Paige, Jonathan S. Ostroff, and Phillip J. Brooke. Principles for modeling language design. *Information and Software Technology*, 42(10):665–675, 2000.

[QGW10] Hazem Qattous, Philip Gray, and Ray Welland. An empirical study of specification by example in a software engineering tool. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. Article No. 16. ACM, 2010.

[RKPP09] Louis M. Rose, Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. Enhanced automation for managing model and metamodel inconsistency. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 545–549. IEEE Computer Society, 2009.

[Rup14] Chris Rupp. *Requirements-Engineering und -Management: Aus der Praxis von klassisch bis agil*. Carl Hanser Verlag GmbH & Co. KG, 2014.

[RWLN89] Jeff Rothenberg, Lawrence E. Widman, Kenneth A. Loparo, and Norman R. Nielsen. The nature of modeling. In *Artificial Intelligence, Simulation and Modeling*, pages 75–92. John Wiley & Sons, 1989.

[SA13] Andreas Scharf and Till Amma. Dynamic injection of sketching features into GEF based diagram editors. In *Proceedings of the 35th International Conference on Software Engineering (ICSE 2013)*, pages 822–831. IEEE Press, 2013.

[SB10] Ugo Braga Sangiorgi and Simone D.J. Barbosa. Sketch: Modeling using freehand drawing in Eclipse graphical editors. In *ICSE Workshop on Flexible Modeling Tools*, 2010. Available online [last checked 07/04/16]: `http://www.academia.edu/369660/SKETCH_Modeling_Using_Freehand_Drawing_In_Eclipse_Graphical_Editors`.

[SBV12] Ugo Braga Sangiorgi, François Beuvens, and Jean Vanderdonckt. User interface design by collaborative sketching. In *Proceedings of the 2012 ACM Conference on Designing Interactive Systems (DIS)*, pages 378–387. ACM, 2012.

[SCDLG12] Jesús Sánchez-Cuadrado, Juan De Lara, and Esther Guerra. Bottom-up meta-modelling: An interactive approach. In *Proceedings of the 15th International Conference on Model-Driven Engineering Languages and Systems (MODELS 2012)*, pages 3–19. Springer-Verlag, 2012.

[SGM10] Norbert Seyff, Florian Graf, and Neil Maiden. Using mobile RE tools to give end-users their own

voice. In *Proceedings of the 18th IEEE Internationl Requirements Engineering Conference (RE 2010)*, pages 37–46. IEEE Computer Society, 2010.

[SKT11] Christian Schäfer, Thomas Kuhn, and Mario Trapp. A pattern-based approach to DSL development. In *Proceedings of the Compilation of the Co-located Workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11 (SPLASH 2011 Workshops)*, pages 39–46. ACM, 2011.

[SNH+09] Patrick C. Shih, David H. Nguyen, Sen H. Hirano, David F. Redmiles, and Gillian R. Hayes. GroupMind: Supporting idea generation through a collaborative mind-mapping tool. In *Proceedings of the 2009 ACM International Conference on Supporting Group Work (GROUP)*, pages 139–148. ACM, 2009.

[Sta73] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, 1973.

[Sut63] Ivan E. Sutherland. *Sketchpad: a man-machine graphical communication system*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering, 1963.

[Tan91] John C. Tang. Findings from observational studies of collaborative work. *International Journal of Man-Machine Studies*, 34(2):143–160, 1991.

[TK09] Juha-Pekka Tolvanen and Steven Kelly. MetaEdit+: Defining and using integrated domain-specific modeling languages. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object-Oriented Programming Systems Languages and Applications (OOPSLA 2009)*, pages 819–820. ACM, 2009.

[Tve02] Barbara Tversky. What do sketches say about thinking? In *Proceedings of the 2002 AAAI Spring Symposium on sketch understanding*. AAAI Press, 2002.

[VdL02] Remko Van der Lugt. Functions of sketching in design idea generation meetings. In *Proceedings of the 4th Conference on Creativity & Cognition (C&C 2002)*, pages 72–79. ACM, 2002.

[VZJ11] Bernhard Volz, Michael Zeising, and Stefan Jablonski. The open meta modeling environment. In *ICSE Workshop on Flexible Modeling Tools*, 2011. Available online [last checked 07/04/16]: `http://www.researchgate.net/publication/266871141_The_Open_Meta_Modeling_Environment`.

[Wac07] Guido Wachsmuth. Metamodel adaptation and model co-adaptation. In *Proceedings of the 21th European Conference on Object-Oriented Programming (ECOOP 2007)*, volume 4609 of *Lecture Notes in Computer Science*, pages 600–624. Springer Berlin Heidelberg, 2007.

[WG11] Dustin Wüest and Martin Glinz. Flexible sketch-based requirements modeling. In Daniel Berry and Xavier Franch, (Eds.): *Requirements Engineering: Foundation for Software Quality*, volume 6606 of *Lecture Notes in Computer Science*, pages 100–105. Springer Berlin / Heidelberg, 2011.

[WH06] Roel J. Wieringa and Hans Heerkens. The methodological soundness of requirements engineering papers: A conceptual framework and two case studies. *Requirements Engineering*, 11(4):295–307, August 2006.

[WHD+11] Jagoda Walny, Jonathan Haber, Marian Dörk, Jonathan Sillito, and M. Sheelagh T. Carpendale. Follow that sketch: Lifecycles of diagrams and sketches in software development. In *Proceedings of the 6th IEEE International Workshop on Visualizing Software for Unterstanding and Analysis (VISSOFT 2011)*, pages 1–8. IEEE, 2011.

[Wou13] Laurent Wouters. Towards the notation-driven development of DSMLs. In Ana Moreira, Bernhard Schätz, Jeff Gray, Antonio Vallecillo, and Peter Clarke, (Eds.): *Model-Driven Engineering Languages and Systems*, volume 8107 of *Lecture Notes in Computer Science*, pages 522–537. Springer Berlin Heidelberg, 2013.

[WSG12] Dustin Wüest, Norbert Seyff, and Martin Glinz. Flexible, lightweight requirements modeling with FlexiSketch. In *Proceedings of the 20th International Requirements Engineering Conference (RE 2012)*, pages 323–324. IEEE, 2012.

[WSG13a] Dustin Wüest, Norbert Seyff, and Martin Glinz. FlexiSketch: A mobile sketching tool for software modeling. In *Proceedings of the 4th International Conference on Mobile Computing, Applications and Services (MobiCASE 2012)*, pages 225–244. Springer, 2013.

[WSG13b] Dustin Wüest, Norbert Seyff, and Martin Glinz. Semi-automatic generation of metamodels from model sketches. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE 2013)*, pages 664–669. IEEE, 2013.

[WSG15a] Dustin Wüest, Norbert Seyff, and Martin Glinz. FLEXISKETCH TEAM: Collaborative sketching and notation creation on the fly. In *Proceedings of the 37th International Conference on Software Engineering (ICSE 2015)*, pages 685–688. IEEE Press, 2015.

[WSG15b] Dustin Wüest, Norbert Seyff, and Martin Glinz. Sketching and notation creation with FlexiSketch

Team: Evaluating a new means for collaborative requirements elicitation. In *Proceedings of the 23rd IEEE International Requirements Engineering Conference (RE 2015)*, pages 186–195. IEEE, 2015.

[WSN+11] Nadir Weibel, Beat Signer, Moira C. Norrie, Hermann Hofstetter, Hans-Christian Jetter, and Harald Reiterer. PaperSketch: A paper-digital collaborative remote sketching tool. In *Proceedings of the 2011 ACM International Conference on Intelligent User Interfaces (IUI)*, pages 155–164. ACM, 2011.

[Wüe11] Dustin Wüest. Bridging the gap between requirements sketches and semi-formal models. In *Doctoral Symposium of the 19th IEEE International Requirements Engineering Conference (RE 2011)*, 2011. Available online [last checked 12/05/15]: `http://www.zora.uzh.ch/55675/1/20120117132036_merlin-id_3715.pdf`.

Appendix A
Publications

This appendix presents the list of publications on which this cumulative dissertation is built. Publications that are not included in the dissertation are marked as such.

A.1 Conference Papers

[WSG15b] Dustin Wüest, Norbert Seyff, and Martin Glinz. Sketching and Notation Creation with FlexiSketch Team: Evaluating a New Means for Collaborative Requirements Elicitation. In *Proceedings of the 23rd IEEE International Requirements Engineering Conference*, pages 186–195, 2015.

[WSG15a] Dustin Wüest, Norbert Seyff, and Martin Glinz. FLEXISKETCH TEAM: Collaborative Sketching and Notation Creation on the Fly. In *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering*, pages 685–688, 2015.

[WSG13b] Dustin Wüest, Norbert Seyff, and Martin Glinz. Semi-automatic generation of metamodels from model sketches. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, pages 664–669, 2013.

[WSG13a] Dustin Wüest, Norbert Seyff, and Martin Glinz. FlexiSketch: A Mobile Sketching Tool for Software Modeling. In *Proceedings of the 4th International Conference on Mobile Computing, Applications, and Services (MobiCASE 2012*, pages 225–244, 2013.

[WSG12] Dustin Wüest, Norbert Seyff, and Martin Glinz. Flexible, lightweight requirements modeling with FlexiSketch. In *Proceedings of the 20th IEEE International Requirements Engineering Conference*, pages 323–324, 2012. (**Not included in this thesis.**)

[WG11] Dustin Wüest and Martin Glinz. Flexible Sketch-Based Requirements Modeling. In *Proceedings of the 17th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 100–105, 2011. (**Not included in this thesis.**)

A.2 Journal articles

[WSG15c] Dustin Wüest, Norbert Seyff, and Martin Glinz. FlexiSketch: A Lightweight Sketching and Metamodeling Approach for End-Users. Software and Systems Modeling, 2015. Under review.

[WSG13c] Dustin Wüest, Norbert Seyff, and Martin Glinz. Von der Idee zum Anforderungsmodell ohne Medienbruch. Extended abstract, in *Software Technik Trends*, Vol. 33 (1), GI, pages 21–22, 2013. (**Not included in this thesis.**)

[WSG15d] Dustin Wüest, Norbert Seyff, and Martin Glinz. Kollaboratives, leichtgewichtiges Erzeugen von Modellskizzen und zugehörigen Notationen im Rahmen von RE Workshops. Extended abstract, accepted 2015, to be published in *Software Technik Trends*, GI. (**Not included in this thesis.**)

A.3 PhD Symposium

[Wüe11] Dustin Wüest. Bridging the Gap Between Requirements Sketches and Semi-Formal Models. In *Doctoral Symposium of the 19th IEEE International Requirements Engineering Conference*, 2011. (**Not included in this thesis.**)

A.4 Technical Reports

[Gli10] Martin Glinz and Dustin Wüest. A Vision of an Ultralightweight Requirements Modeling Language. Technical Report No. IFI-2010.0006, Version: 1, 2010. (**Not included in this thesis.**)

Curriculum Vitae

Name:	Dustin Wüest
Date of Birth:	September 22, 1982
Place of Citizenship	Lucerne, LU, Switzerland

2009 - 2015	Assistant and doctoral student at the *University of Zurich*, ZH, Switzerland
2007 - 2009	Master in Computer Science (Multimodal and Cognitive Systems) at the *University of Zurich*, ZH, Switzerland
2004 - 2007	Bachelor in Computer Science (Information Systems) at the *University of Zurich*, ZH, Switzerland
2001 - 2004	Studies in Computer Science and Physics at the *Swiss Federal Institute of Technology (ETH) Zurich*, ZH, Switzerland
1995 - 2001	High school, Gymnasium Immensee, SZ, Switzerland